A Performance Evaluation and Benchmarking Tool

by

Atul Kumar

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

JANUARY, 1998

A Performance Evaluation and Benchmarking Tool

A Thesis Submitted
in Partial Fulfillment of the Requirements
for the Degree of
Master of Technology

by
Atul Kumar

to the

Department of Computer Science & Engineering
Indian Institute of Technology, Kanpur
January, 1998

CSE-1998-M-KUM-PER

Entered in System

[illegible]
4 6.98

Certificate

Certified that the work contained in the thesis entitled "A Performance Evaluation and Benchmarking Tool", by Mr.Atul Kumar, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

[illegible]

(Dr. Harish Karnick)
Professor,
Department of Computer Science & Engineering,
Indian Institute of Technology,
Kanpur.

December, 1997

Abstract

Computer system designers, administrators, and users are interested in performance evaluation since their goal is to obtain or provide the maximum performance at lowest cost. System performance varies enormously from one application domain to another, therefore, no single metric can measure the performance of computer systems for all applications. Load on various components of the system also affect the performance considerably. Several performance evaluation and benchmarking tools exist, which measure the performance of a computer system for a particular workload. However, very few benchmarks, if any, check the performance of a system for varying system loads. Most of the benchmarks are designed to run on an 'idle system'. Therefore, they present a measure of 'peak performance' of a system for a particular type of workload. In real life, these 'peak performance' figures are of little help.

This tool measures the load on different components of a system and evaluates the system performance by running different synthesized test programs. These programs have been selected to approximate the workload of the IIT Kanpur computing environment. The results of performance evaluation have been presented along with the different components of current system load.

Acknowledgments

On the outset I want to thank **Dr. Harish Karnick** for his support and guidance throughout my work. In spite of his various involvements he was always available and patient. My thanks to **Mrs. Karnick** for being extremely affectionate. I wish to thank my family for encouraging me to go for post-graduate studies. I would also like to thank my friends Ambarish, Kshitiz, Shyam and Vihari who helped me with lots of ideas during the design phase and then during the writing and typesetting of this report. My thanks to my friends Manoj, Santosh, Vivek, Avi, and Suman with whom I shared some unforgettable moments during my stay at IITK.

My gratitude goes to all of my batch-mates, who made my stay here, in the lab as well as in IITK, a memorable one.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the early days of computing, a programmer's main goal was to get a working program with little thought about its efficiency. In 1946 Von Neumann compared the speed with which the early computers (including ENIAC) performed multiplication when computing ballistic trajectories[McK88]. Herbst *et al.*, in 1955, measured the instruction mix of programs running on the Maniac computer.

Computer system designers, administrators, and users are all interested in performance evaluation. Designers a compare number of alternative designs to find the best design. Administrators compare a number of alternative systems to decide the best system for a set of applications. Users compare a number of installed systems to find the best system for a particular job. To get maximum performance at lowest cost is the key criterion in design, procurement, and use of computer system.

The performance evaluation of a computer system is an ambiguous task. Various people may think of entirely different things when they use the term "performance". The people concerned with the use of large databases tend to think of performance as the number of transactions performed per second while people from the scientific and technological areas may be interested in the number of floating point operations per second. Even with a narrow focus the assessment of performance is not straightforward. Suppose one is interested in only scientific computing. A large variety of high performance computers are available for scientific computing, varying from vector computers with a limited number of processors sharing common memory, to machines with thousands of processors and distributed memories. The performance range of these machines can vary, sometimes by a factor of thousand or more, depending on how the problem and the program suit the underlying architecture and operating system.

1.1 Goals of Performance Evaluation

Goals are an important part of all endeavors. Any endeavor without goals is bound to fail. Performance evaluation projects are no exception. [Jai91] presents a list of common mistakes that can be found in identifying the goals of a performance evaluation project. Design of a performance evaluation tool or benchmark needs to identify what exactly one is trying measure. The common targets for measuring performance are CPU and disk I/O. With the current trend towards disk-less or data-less[1] workstations, the performance of the file server and the network has become crucial and so has performance measurement.

Performance measurement tools are normally used to predict the performance of an unknown system on a known, or at least well defined task or set of tasks. The performance results are used to make purchase decisions for a new system. Such tools can also be used as monitoring and diagnostic tools. One can potentially pinpoint the cause of poor performance by running a test program and comparing the results against a known configuration. Similarly, one can run a test program after making a change to determine the improvement or degradation in performance.

The best program to test a system's performance is the actual application that will run on the system. But that is not always possible because the applications are not always ready before the system is purchased. Even if the application is available before purchasing the system, the same computer system can perform differently on the different runs of the same application as the input data and other parameters may vary. Another problem is that very few systems are dedicated to a particular task and most of the installations are used for running various applications. If the nature of the jobs that different applications are performing is the same then the test programs

[1]Data-less workstations are the workstations with small capacity local disk. This disk contains the OS files and binaries but not the user files

can be designed to predict the system performance for those type of jobs. But in the hybrid environments present in educational institutions. the same resources are used for various applications (unless the application is very specific). Therefore, no single number can represent system performance, rather the performance of different subsystems (eg CPU, disk I/O etc) is measured and the result is presented as a set of numbers.

1.2 Workload Selection and Benchmarking

Every measure of performance requires some specification of the workload that is to be handled. Practical considerations require that only a few of the many possible jobs be selected as representative of the work expected from the system. Some criteria are:

- Jobs that run most frequently
- Jobs that account for most of the system's time
- Jobs with critical response (completion) time requirements

Understanding the workload is the foundation for performance studies. Once the workload is understood, evaluators can design a sequence of tests to investigate specific components of performance, culminating in tests that correlate directly with a given workload. Workload parameters include job CPU time, job I/O requests, I/O service time, job priority, job memory usage and job sleeping time. Values for these parameters are selected, such that they can closely approximate the actual system load.

A **benchmark** is a documented procedure that measures the time needed by a computer system to execute well defined computing task. It is assumed that this time is related to the performance of the computer system and that somehow the same procedure can be applied to other systems so that comparisons can be made between different hardware/software configurations. [NB75] presents a detailed discussion on 'benchmarking'. [Don87] discusses some pitfalls in computer benchmarking. Before starting benchmark design (or selection), a basic choice must be made between **synthetic benchmarks** and **applications benchmarks**.

1.2.1 Synthetic Vs Application Benchmarks

Synthetic benchmarks[Gaz] are specially designed to measure the performance of individual components of a computer system, usually by exercising the chosen component to its maximum capacity. For example **Whetstone** is a synthetic benchmark used to measure the floating point performance of a CPU. They should measure a specific aspect of the system being tested, independent of all other aspects. For example a synthetic benchmark for Ethernet card I/O throughput should result in the same or similar figures whether it is run on a 386SX-16 with 4 MBytes of RAM or a Pentium 200 MMX with 64 MBytes of RAM.

Application benchmarks try to measure the performance of computer systems for some category of real-world computing tasks. A commonly executed application is chosen and the time to execute this application is used as a benchmark.

1.2.2 Low-level Vs High-Level Benchmarks

Low-level benchmarks[HOW] directly measure the performance of the hardware: CPU clock, DRAM and cache SRAM cycle times, hard disk average access time, latency and track-to-track stepping time, etc... Such benchmarks are used to verify the figures given in the data sheet of a component. Another use of low-level benchmarks is to check whether a kernel driver is correctly configured for a specific piece of hardware.

High level benchmarks are more concerned with the performance of the hardware-driver-OS-compiler combination for a specific aspect of the computer system. For example file I/O performance, or even for a specific hardware-driver-OS-compiler-application performance, e.g. benchmarking a specific Web server package on different microcomputer systems, or different Web server packages on the same system.

1.3 Motivation

Several benchmarks exist for measuring the performance of different aspects of computer systems. All these benchmarks produce one or more numbers and these numbers are presented as the measure of system performance. A well known computer system is considered as the reference machine and the results of running these benchmarks on the reference machine are published so that one can compare the performance of a machine with that of the reference machine.

IIT Kanpur has a number of computer systems which enjoy a high rating on the popular benchmarks. But the system performance obtained in the real environment is not very good. The performance of these systems is not up to expectations. The high load on these systems is considered the main reason for low performance. Saying that the systems are overloaded is not sufficient since the overall system consists of several components and it is hard to find the bottleneck.

The IITK environment consists of several servers connected by a TCP/IP Network. These servers are accessed through dumb terminals. X-terminals, and disk-less or disk-full workstations. Dumb terminals are attached through terminal servers. X-terminals and Workstations run the X-Windows system. Window manager and other X-applications run either locally or on one of the servers. User files are served on different machines by the Network File Service running on the servers. The workload varies from the "edit-compile-execute-debug" cycle to highly CPU bound jobs such as programs solving differential equations. or computing Fourier transforms, or performing multiplication and other operations on large matrices. Some users run a mix of CPU and I/O bound jobs such as programs simulating an assembly line of a production unit.

Running a benchmark designed for a specific workload in such an environment is not useful. The need is for a way to relate the current system load (load on different subsystems) to its performance. Unfortunately, very little work has been done on this. Our primary motivation is to provide a tool which can measure the performance of different subsystems (eg. CPU, disk I/O, NFS, Network) along with the current load on different subsystems. The result of running a benchmark for measuring the performance of a subsystem should be listed with the current system loads which

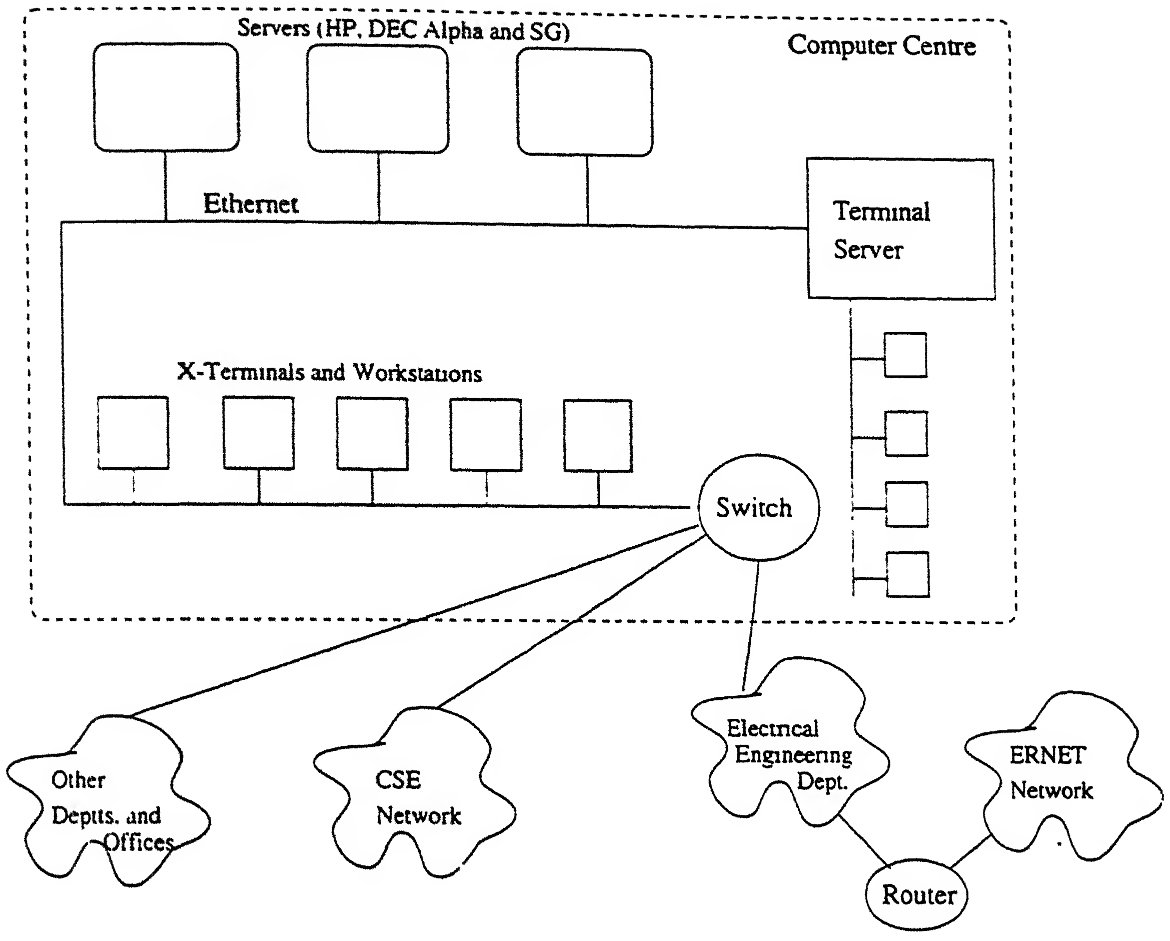

Servers (HP, DEC Alpha and SG)
Computer Centre
Ethernet
Terminal Server
X-Terminals and Workstations
Switch
Other Deptts. and Offices
CSE Network
Electrical Engineering Dept.
ERNET Network
Router

Figure 1: IITK Computing Resources

can affect the performance of that subsystem. The program should run periodically on a working system and record system performance with real-life load. For the new systems the tool should generate artificial random load (that closely approximates the real-life load) and measure the system performance.

1.4 Organization of the Thesis

First we discuss some popular benchmarks and the workload they assume. Then we discuss the scope of this performance evaluation tool. After this we present the design of our performance evaluation tool. This includes the techniques used for collecting system parameters and system load. and the workload different tests generate. In results. we present the results of running this tool on the different machines in the IITK Computer Center and Computer Science and Engineering Department. In conclusions, we will see further enhancements that can be made in the scope of the performance evaluation tool.

Chapter 2

An Overview of Benchmarks

2.1 Introduction

One of the oldest measure of CPU speed is MHz, i.e., the processor clock frequency in Mega Hertz. This measure is not useful with today's wide range of CPUs as the word length and architecture of the CPU matters a lot in determining the CPU speed. Another popular, though superficial, measure of CPU speed is MIPS (Millions of Instructions Per Second). On a family of systems with the same processor, the MIPS rating can help judge relative system integer performance. The ultimate apple-to-orange phenomenon occurs when we use the MIPS ratings to compare two different architectures like RISC and CISC (Reduced and Complex Instruction Set Computer). To make sense of MIPS rating users must define the term *instruction*. A direct correlation does not exist between the number of instructions being executed and the amount of actual work being accomplished. In the world of super computers, the traditional unit is MFLOPS (Millions of Floating Point Operations Per Second). This often means 'Peak MFLOPS' the highest rate of floating point operations per second, obtainable only in ideal cases.

Any attempt to give MIPS numbers some useful meaning boils down to running a representative program or set of programs. Therefore, it is better to drop the notion of MIPS and just measure the speed of these *b*enchmark programs.

This chapter presents an overview of commonly available benchmarks. These benchmarks are divided into five major categories: CPU performance benchmarks, disk I/O and NFS performance benchmarks, Network performance benchmarks, Overall Performance Benchmarks, and Miscellaneous benchmarks.

2.2 CPU Performance Benchmarks

2.2.1 General purpose CPU benchmarks

- *Whetstone*

The Whetstone benchmark[Wei91, Pri89] was the first program in the literature that was explicitly designed for benchmarking purposes. Its authors are H.J. Curnow and B.A. Wichmann from the National Physical Laboratory in Great Britain. It was published in 1976, with ALGOL 60 as the publication language. Today it is almost exclusively used in its FORTRAN version, with either single precision or double precision for floating-point numbers.

The Whetstone benchmark owes its name to the Whetstone ALGOL compiler system. This system was used to collect statistics about the distribution of 'Whetstone instructions', instruction for the intermediate language used by this compiler, for a large number of numerical programs. A synthetic program was then designed, consisting of several modules where each module contains statements of some particular type (e.g. integer arithmetic, floating-point arithmetic, if statements, calls), ending with a statement printing the results. Weights are attached to the different modules (realized as loop bounds for loops around the individual modules' statements) such that the distribution of Whetstone instructions for the synthetic benchmark matches the distribution observed in the program sample. The weights have been chosen in a way that the program executes a multiple of one million of these Whetstone instructions; benchmark results are given accordingly as KWIPS (Kilo Whetstone Instructions Per Second) or MWIPS (Mega Whetstone Instructions Per Second). Whetstone has a high percentage of floating-point operations. A large fraction of the execution time is spent in mathematical library functions.

▪ *Dhrystone*

As the name already indicates, Dhrystone[Wei91, Pri89, Ser] was developed in a similar way to Whetstone, it is a synthetic benchmark published by the Reinhold Weicker and Siemens Nixdorf in 1984. The original language of publication is Ada, although it uses only the 'Pascal subset' of Ada and was intended to be easily translatable to Pascal and C; presently it is mainly used in the C version.

Dhrystone is based on a literature survey on the distribution of source language features in non-numeric, system-type programming (operating systems, compilers, editors, etc.). It has been observed that in addition to the obvious difference in data types (integral types vs. floating point types), numeric and system-type programs have other differences also: System programs contain less loops, simpler computational statements, more 'if' statements and procedure calls.

Dhrystone consists of 12 procedures, they are included in one measurement loop with 94 statements. During one loop ('one Dhrystone'), 101 statements are executed dynamically. The results are usually given in 'Dhrystones per second'. Dhrystone contains no floating-point operations in its measurement loop. A considerable percentage of execution time is spent in string functions. In extreme cases this number goes up to 40%. Dhrystone is very sensitive to optimization and results will appear erratic unless degree of optimization is carefully tracked.

▪ *Digital Review*

*D*igital Review magazine[Pri89] has compiled a set of benchmark routines that mixes 34 individual integer and floating-point routines. The test itself stresses floating-point performance. This large benchmark contains over 3,000 lines of FORTRAN code. The Digital Review benchmark does not perform any verification of test results; these results usually appear as a list of a geometric mean of all tests performed (in seconds). At a secondary level the test normalizes relative comparisons among various systems to the Digital MicroVAX II, which is taken as 1.0. These units are called MicroVAX units of processing (MVUPs).

Users have criticized this benchmark for its odd structure and unusual instruction mix that does not accurately represents the real-world program flow. Initializing the routines within the timing loops, rather than running the actual benchmark code, consumes a large amount of time. Digital Review magazine has taken steps to revise its benchmark and the new version is now called CPU2. CPU2 is a floating-point intensive series of FORTRAN programs and consists of thirty-four separate tests. The benchmark is most relevant in predicting the performance of engineering and scientific applications.

2.2.2 Integer Only benchmarks

▮ *Sim*

The SIM[FAQ, Ser] program finds k best non-intersecting alignments between two sequences or within one sequence. The program is based on an algorithm presented by Xiaoqiu Huang and Webb Miller of the Pennsylvania State University

Using dynamic programming techniques, SIM is guaranteed to find optimal alignments. The alignments are reported in order of similarity score, with the highest scoring alignment first. The k best alignments share no aligned pairs. SIM requires space proportional to the sum of the input sequence lengths and the output alignment lengths. Thus SIM can handle sequences of tens of thousands, or even hundreds of thousands, of base pairs on a workstation.

▮ *Fhourstones*

Fhourstones[Ser, FAQ] is a small integer-only program that solves positions in the game of connect-4 using exhaustive search with a very large transposition table. The program is written in C.

▮ *Heapsort*

Heapsort[Ser, FAQ] is an integer program that uses the "heap sort" method of sorting a random array of long integers up to 2 megabytes in size. Benchmark result is given as MIPS rating, based upon the program run time for one iteration. A gcc (GNU C Compiler) 2.1 unoptimized assembly dump count of instructions per iteration for a i486 machine is taken as the reference. A heapsort MIPS rating is not representative of a 'typical' instruction mix.

■ *Hanoi*

Hanoi[Ser. FAQ] is an integer program that solves the 'Towers of Hanoi' puzzle using recursive function calls.

2.2.3 Scientific Benchmarks

■ *Linpack*

As described by its authors. Linpack[MBL91. Pri89. Wei91] didn't originate as a benchmark. it was first just a collection (a package. hence the name) of linear algebra subroutines often used in FORTRAN programs. It was first published in 1976 by Jack Dongarra of the University of Tennessee. The program operates on a large matrix (2-dimensional array): however. the inner subroutines manipulate the matrix as a one-dimensional array. The matrix size is 100x100.

The results are usually reported in terms of MFLOPS. the number of floating-point operations executed by the program can be derived from the array size. This terminology implies that the non-floating point operations are neglected or, stated otherwise. that their execution time is included in that of the floating-point operations. Linpack has a high percentage of floating point operations. However, only a few floating-point operations are actually used. For example, there are no floating-point divisions in the program. and no mathematical functions are used at all. The execution time is almost exclusively spent in one small function. This means that even a small instruction cache will show a very high hit rate.

The Java version of Linpack is also available now[Ver].

▪ *Livermore Fortran Kernels*

The 'Livermore Fortran Kernels[MBL91], also called the 'Lawrence Livermore Lops', consist of 24 'kernels' i.e. inner loops of numeric computations from different areas of the physical sciences. The author, F.H. McMahon (Lawrence Livermore National Laboratory, Livermore), has collected them into a benchmark suite and has added statements for time measurement. The individual loops range from a few lines to about one page of source code. The program is self-measuring and computes MFLOPS rates for each kernel, for three different vector lengths. These kernels contain a high amount of floating-point computations and a high percentage of array accesses.

The performance metric reported for the Livermore Fortran Kernels is the MFLOPS rate for each individual kernel along with the overall average, harmonic, and geometric means for the entire suite. For vector machines, an all scalar-compilation run is required to measure the basic scalar performance range of the processor.

▪ *NAS kernel benchmark*

The NAS (Numerical Aerodynamic Simulation)[MBL91] Kernel Benchmark was developed at the NASA Ames Research Center. It consists of seven test kernels. Each individual test kernel consists of a loop that iteratively calls a particular subroutine. These subroutines were supported by a number of NASA Ames scientists and programmers involved with computational fluid dynamics projects. All the seven selected programs emphasize the vector performance of a computer system. In fact, almost all of the floating-point operations are contained in the loops that are computable by vector operations. Input data array were generated using a portable pseudo random number generator. Each of the kernels is independent from the others, i.e. none depends on the results calculated in a previous test program.

▪ *EDN Benchmarks*

The EDN Benchmarks[Wei91] were developed by a group at CMU for the project 'Military Computer Family', it was published by EDN in 1981. Originally, the programs were written in several assembly languages (LSI-11/23, 8086, 68000, Z8000); the intention was to measure the speed of the microprocessor without also measuring the compiler's quality. The benchmark programs include programs for string search, bit test/set/reset, linked list insertion, quicksort, and bit matrix transformation. The C version of the benchmarks is also available but that is not standard and the programs are disseminated in an informal way only.

▪ *SPICE*

The general-purpose SPICE (Simulation Program with Integrated Circuit Emphasis) [Pri89] came from the University of California at Berkeley. This benchmark makes heavy use of both integer and double-precision, floating-point calculations (the floating-point operations are not vector oriented). Because it is quite large, the program is a good test of system instruction and data-cache performance. SPICE accepts a circuit description as input and simulates the design. The user can monitor currents and voltages at various circuit locations. UC Berkeley and several system vendors have distributed various input data packs for simulation of different types of circuits.

▪ *Stanford Integer and Floating Point Benchmarks*

The **Stanford Integer**[Wei91, Pri89] benchmark is written in the C programming language. The suite consists of small programs that use algorithms to solve real-world problems. Some of these small programs include the Towers of Hanoi and the Eight Queens puzzles, multiplication of integer matrices, and the quick and bubble sorts. Yet another routine inserts and recursively searches a binary tree. Test measurements consist of the geometric mean of all results.

Stanford Floating Point[Wei91, Pri89] benchmark uses tight loops and a large proportion of floating-point code. The routine's susceptibility to code optimization by high-quality compilers affects the results. The suite consists of the fast Fourier transform (FFT) and matrix multiplication (MM) tests. The first test typically computes a 256-point, single-precision FFT 20 times. The second test multiplies two 40x40 single-precision matrices.

▪ *Los Alamos Benchmarks*

The Los Alamos National Laboratory (LANL) maintains a set of 13 portable FORTRAN-77 benchmark[MBL91] programs that represent the Laboratory workload. These benchmarks are intended to represent the types of algorithms of current importance to LANL as well as the characteristic style of coding found within actual production codes run at the Laboratory. The LANL benchmark set consists of a hierarchy of codes: hardware demonstration kernels, basic routines, and stripped down application programs since they come closest to representing the true LANL workload.

Two of the original benchmark programs measure rates in MFLOPS for primitive vector operations as a function of vector length. The MFLOPS extracted using these benchmarks normally over represent the computational power. than the actual results which can be obtained for a real application code.

▪ *Sieve of Eratosthenes*

One of the most popular programs for benchmarking in the world of small PC's is the 'Sieve of Eratosthenes'[Wei91], sometimes also called 'Primes'. It computes all prime numbers up to a given limit (usually 8192). The program has some unusual characteristics: 33% of the dynamically executed statements are assignments of a constant, only 5% are assignments with an expression at the right hand side. There are no 'while' statements and no procedure calls, 50% of the statements are loop control evaluations. All operands are integer operands, 58% of them are local variables.

▪ *Dodoc*

This is a 5,300 line FORTRAN program[Pri89], which simulates the operations within a nuclear reactor. This program accurately tests instruction-fetch bandwidth and scalar floating-point performance. Compilers can vectorize very little of the code. The routine uses the Monto Carlo method of simulation in which an iterative process converges on an expected result. The routine was originally designed as a check of both compiler and real-world functions. Normalized results appear in terms of the ratio of CPU time needed to perform the test versus an arbitrarily defined reference. This R factor is normalized where 100 equals the performance of the IBM 370 Model 168. The expression is: R=48,671/seconds of processor time. Larger R factors mean to higher system performance.

2.2.4 High Performance Scientific Computing

▪ *EuroBen Group Benchmarks*

The EuroBen[vdS91] group was established in mid-1990 by a group of people that was concerned about obtaining the performance profile of high-performance scientific computers. As the founders of EuroBen believe that characterization of the performance for high-performance scientific computers cannot be done by a single performance measure, especially where vector and parallel architectures are involved. A graded approach was used to ensure a more general assessment of the performance. These program range from very simple to complete algorithms that are important in certain application areas. The simple programs give basic information on the performance of operations, intrinsic functions, etc., which can do much to identify the strong and weak points of a machine. When one wants to extract more information one should conduct the next level of tests which contains simple but frequently used algorithms like FFTs, random number generation, etc. When one wants more information one has to run more modules. All the modules are written in FORTRAN-77 and the precision required is at least 64 bits.

▪ *The Perfect Benchmark*

Perfect (Performance Evaluation for Cost-Effective Transformations)[MBL91] benchmarking activity was started in 1987 by a group of academic and industrial collaborators. The original ambitious goals were to define an applications-based methodology for supercomputer performance evaluation and in the process assemble and port a suite of scientifically relevant codes to numerous high performance computing machines. The resulting set of codes, known as the Perfect benchmarks, consists of 13 programs drawn from a variety of scientific and engineering fields. With over 60,000 lines of FORTRAN source listing, several of these programs have been successfully ported to over 30 machines.

- *Flops*

Flops[FAQ, Ser] estimates the MFLOPS rating for specific floating-point addition (FADD), floating-point subtraction (FSUB), floating-point multiplication (FMUL), and floating-point division (FDIV) instruction mixes. Four distinct MFLOPS ratings are provided based on the FDIV weightings from 25% to 0% and using register-to-register operations. This benchmark works with both scalar and vector machines.

- *STREAM*

STREAM[FAQ] is a synthetic benchmark which measures sustainable memory bandwidth with and without simple arithmetic, based on the timing of long vector operations. STREAM is available in FORTRAN and C versions, and the results are used by all major vendors in high performance computing.

2.3 Disk I/O and NFS Performance Benchmarks

2.3.1 IOBENCHP

IOBENCHP[FAQ] was written by Barry Wolman of Prime Computer. IOBENCHP is a multi-stream benchmark that uses a controlling process (iobench) to start, coordinate, and measure a number of 'user' processes (iouser).

There are four workloads associated with the IOBENCH: short, ref, test, and elapsed. The 'short' workload takes only a few seconds to run and just verifies that IOBENCHP was made properly. The two 2MB files used by the 'short' workload are created automatically in the result directory. The 'ref' workload uses four 20MB files with 16, 32, 48, 64, and 80 users using record lengths of 100 bytes, 4096 bytes, and 8192 bytes. For each run, 10,000 'cycles' are spread across the number of users. The 'ref' workload should take about 5-20 minutes to run depending on number of disk controllers and type and number of disks used. The 'test' workload is the same as the 'ref' workload, except no mounting is done. The 'elapsed' workload uses the same four 20MB files and uses 50,000 'cycles' spread across 50 users with a 4096 byte record length. The elapsed workload will take at least 5 minutes. Both the 'ref' and 'elapsed' workloads require that mountable file systems be specified: these will be unmounted (if necessary) before the run, mounted, and then left unmounted at the end of the run.

2.3.2 IOZONE

This test[FAQ] writes a X MEGABYTE sequential file in Y byte chunks, then rewinds it and reads it back. The size of the file should be big enough to factor out the effect of any disk cache.

The file is written (filling any cache buffers), and then read. If the cache is greater than X MB, then most if not all the reads will be satisfied from the cache. However, if it is less than or equal to half of X MB, then NONE of the reads will be satisfied from the cache. This is because after the file is written, a X/2 MB cache will contain the upper half of the test file, but the program will start reading from the beginning of the file (data which is no longer in the cache). In order for this to be a fair test, the length of the test file must be AT LEAST 2X the amount of disk cache memory in the system. If not, then one is really testing the speed at which the CPU can read blocks out of the cache (not a fair test)

IOZONE does not normally test the raw I/O speed of the disk or system. It tests the speed of sequential I/O to actual files. Therefore, this measurement factors in the efficiency of the file system, C compiler, and C runtime library. It produces a measurement which is the number of bytes per second that the system can read or write to a file.

2.3.3 NFS Stone

NFS Stone[FAQ] takes two arguments. The first one is the directory to run the tests in. The second one is the lock file name. The program sets itself up, then tries to obtain the lock. This allows the benchmark to sync up. When it has the lock, it releases it right away to allow someone else to get it. This should happen fast enough that the clients will start within a second or two of each other. NFS Stone performs the various operations such as 'make dir', 'remove dir', 'copy', 'move', 'detete dir', 'read file', 'write file', 'load large programs' etc. Different weights are assigned to different operations, and a term 'NFSSTONE operations' is derived using a formula. The benchmark presents the performance as 'NFSSTONEs per second'.

2.3.4 NHFS Stone

This benchmark[FAQ] is intended to measure the performance of file servers that follow the NFS protocol. The work in this area continued within the LADDIS group and finally within SPEC. The SPEC benchmark 097.LADDIS (SFS benchmark suite.is intended to replace Nhfsstone. it is superior to Nhfsstone in several aspects (multi-client capability, less client sensitivity).

Nhfsstone is used on an NFS client to generate an artificial load with a particular mix of NFS operations. It reports the average response time of the server in milli-seconds per call and the load in calls per second. The program adjusts its calling patterns based on the client's kernel NFS statistics and the elapsed time. Load can be generated over a given time or number of NFS calls. Because it uses the kernel NFS statistics to monitor its progress, nhfs- stone cannot be used to measure the performance of non-NFS filesystems. Since it is measuring servers, it should be run on a client that will not limit the generation of NFS requests. This means it should have a fast CPU, lots of memory, a good ethernet interface and the machine should not be used for anything else during testing. Nhfsstone assumes that all NFS calls generated on the client are going to a single server, and that all of the NFS load on that server is due to this client. To make this assumption hold, both the client and server should be as quiescent as possible during tests. If the network is heavily utilized the delays due to collisions may hide any changes in server performance. High error rates on either the client or server can also cause delays due to retransmissions of lost or damaged packets.

2.4 Network Performance Benchmarks

2.4.1 Nettest

Nettest[FAQ] is a network performance analysis tool developed at Cray.

2.4.2 Netperf

Netperf[Hom, Div96] is a benchmark that can be used to measure various aspects of networking performance. Its primary focus is on bulk data transfer and request/response performance using either TCP or UDP and the Berkeley Sockets interface. There are optional tests available to measure the performance of DLPI (Data Link Provider Interface), Unix Domain Sockets, the Fore ATM API and the HP HiPPI LLA interface. Netperf program can perform the following tests:

- TCP Stream Performance
- XTI(X/open Transport Interface) TCP Stream Performance
- UDP Stream Performance
- XTI UDP Stream Performance
- DLPI Connection Oriented Stream Performance
- DLPI Connection Stream
- Unix Domain Stream Sockets
- Unix Domain Datagram Sockets
- Fore ATM API Stream
- TCP Request/Response Performance

- TCP Connect/Request/Response
- XTI TCP Request/Response Performance
- UDP Request/Response Performance
- XTI UDP Request/Response Performance
- DLPI Connection Oriented Request/Response Performance
- DLPI Connection Request/Response Performance
- Unix Domain Stream Socket Request/Response Performance
- Unix Domain Datagram Socket Request/Response Performance
- Fore ATM API Request/Response Performance

2.5 Overall Performance Benchmarks

2.5.1 SPEC Benchmarks

SPEC (Standard Performance Evaluation Corporation)[Dix91, Org] is a new, evolving standard in benchmarking. SPEC, a non profit corporation was founded in 1988 by Apollo, HP, MIPS and Sun Microsystems. SPEC is a suite of different benchmarks meant to measure different aspects of systems performance.

■ *New CPU Benchmarks: SPEC95*

These benchmarks measure the performance of CPU, memory system, and compiler code generation. They normally use UNIX as the portability vehicle, but they have been ported to other OS as well. The percentage of time spent in OS and I/O functions is generally negligible.

CINT 95, integer programs, representing the CPU intensive part of system or commercial application programs.

CFP95, floating point programs. representing the CPU intensive part of numeric scientific application program.

■ *Old CPU benchmarks: SPEC92*

CIN92: this suite contains six benchmarks performing integer computations, all of them written in C.

CIF92: this suite contains 14 benchmarks performing floating point computations. 12 of them are written in FORTRAN and two in C.

■ *SDM Benchmark suite*

SDM stands for 'System Development Multitasking'; the benchmarks in this suite characterize the capacity of a system in a multiuser environment.

■ *SFS Benchmarks Suite*

SFS stands for 'System Level File Server', this is designed to provide a fair consistent and complete method for measuring and reporting NFS performance. SFS Release 1.1 contains one benchmark, 097.LADDIS. This benchmark measures NFS file server performance in terms of NFS response time and throughput. It does this by generating a synthetic NFS workload based on a workload abstraction of an NFS operation mix and an NFS operation request rate.

Running 097.LADDIS requires a file server (the entity being measured) and two or more "load generators" connected to the file server. The load generators are each loaded with 097.LADDIS and perform the 097.LADDIS workload on file systems exported by the file server. The SFS Steering Committee is working on a new version of the SFS benchmark suite. Its main features will be the support for NFS protocol version 3 and to generate a modified workload.

■ *SPEC hpc96 Benchmark Suite*

At the supercomputing'95 conference (Dec'95), the SPEC High Performance Computing Group (HPCG) announced the SPEC hpc96 benchmark suite, consisting of two benchmarks:

SPEC Seis96: an industrial application based on modern seismic processing programs used in search of oil and gas.

SPEC Chem96: an improved version of the program called GAMESS, that come from the US department of Energy's National Resource for Computing in Chemistry.

- *Forthcoming SPEC Benchmarks*

  - *SPECcpu98* - A couple of important issues , besides others, are that the programs should be such that they can be made compute bound and that they can be made portable across difference hardware architectures and operating systems. The emphasis is again going to be on system's processor, memory hierarchy and compiler.

  - *SPECsmt97* - The goal of this benchmark is to have a measure of a System's Multitasking Tasking capabilities. Currently, this benchmark is being created. Work is also being done on developing a standardized set of development tools (e.g. GCC, Perl, GhostScript, etc.) so that it is ensured that each system tested is actually performing the very same amount of work (and is not dependant upon the varying degrees of bells and whistles that a vendor installs).

  - *SPECweb97* - This benchmark will measure thr performance of computers (both h/w and s/w) that are used as www servers. Similar to the SFS benchmark, the benchmark code runs on one or more load generators(clients) that generate HTTP requests over a network(LAN). New issues under consideration are Dynamic Content, Support for Keep Alive and Multiple workloads.

2.5.2 Khornerstone

Developed by Workstation Laboratories, this benchmark[Pri89] yields a normalized rating on overall system performance using 22 separate tests. This suite of tests include a mix of both public-domain and proprietary benchmark routines. The result is a unit of measure called Khornerstones per second. This set of routines measures characteristics of processor, floating-point, and disk performance. The Khornerstone test measures single-user loads on a system and therefore does not accurately measure multiuser performance.

2.5.3 AIM

Aim Technology in Palo Alto[Pri89], CA sells and maintains two suits of multiuser benchmark tests.

Suite III, written in the C programming language, this suite simulates applications that fall into either task or device specific category. The task-specific routines simulate functions such as word processing, database management, and accounting. The device-specific code measures the performance of hardware features like memory, disk, floating-point, and I/O operations. All measurement represent a percentage of VAX 11/780 performance. In general, the AIM Suite III gives an overall performance indication.

Suite V measures throughput in a multitasking workstation environment. The design goals of this new suite includes incremental system loading to gradually increase the stress on system resources, and testing multiple aspects of system performance. The graphically displayed results plot the workload level versus time. Several different models characterize various user environments (financial, publishing, software engineering). The published reports are copyrighted.

2.5.4 SysMark

SysMark93[FAQ] for DOS and WINDOWS was introduced by The Business Applications Performance Corp. (BAPCo), Santa Clara, CA, in 1993. This benchmark software provides objective performance measurement based on the world's most popular PC applications and operating systems.

SYSmark93 provides benchmarks that can be used to objectively measure performance of IBM PC-compatible hardware for the tasks users perform on a regular basis. The benchmarks are comparative tools for those who make purchasing decisions for anywhere from 10 to a thousand or more PCs. SYSmark93 has been endorsed by the BAPCo membership, which includes the world's leading PC hardware and software vendors, chip manufacturers, and industry publications.

SYSmark93 benchmarks represent the workloads of popular programs in such applications as word processing, spreadsheets, database, desktop graphics and software development. Benchmarking can be conducted on the user's own system or at a vendor's site using the standards set by BAPCo to ensure consistency of the results.

2.5.5 Byte Unix Benchmarks

This is a benchmark suite[FAQ] similar in spirit to SPEC, except that it is smaller and contains mostly things like "sieve" and "dhrystone". If one is comparing different Unix machines for performance, this gives fairly good numbers. The suite include programs for measuring arithmetic overhead, system call overhead, filesystem throughput, pipe throughput, etc.

2.6 Miscellaneous Benchmarks

2.6.1 TPC

The TPC is a non-profit corporation founded to define transaction processing and database benchmarks and to disseminate objective, verifiable TPC performance data to the industry. TPC has defined the following benchmarks[TPC, JG91, FAQ]:

- *TPC-A*

TPC-A is a standardization of the Debit/Credit benchmark which was first published in DATAMATION in 1985. It is based on a single, simple, update-intensive transaction which performs three updates and one insert across four tables. Transactions originate from terminals, with a requirement of 100 bytes in and 200 bytes out. There is a fixed scaling between tps rate, terminals, and database size. TPC-A requires an external RTE (remote terminal emulator) to drive the SUT (system under test).

- *TPC-B*

TPC-B uses the same transaction profile and database schema as TPC-A, but eliminates the terminals and reduces the amount of disk capacity which must be priced with the system. TPC-B is significantly easier to run because an RTE is not required.

- *TPC-C*

TPC-C is completely unrelated to either TPC-A or TPC-B. TPC-C tries to model a moderate to complex OLTP system. The benchmark is conceptually based on an order entry system. The database consists of nine tables which contain information on customers, warehouses, districts, orders, items, and stock. The system performs five kinds of transactions: entering a new order, delivering orders, posting customer payments, retrieving a customer's most recent order, and monitoring the inventory level of recently ordered items. Transactions are submitted from terminals providing a full screen user interface. (The specification defines the exact layout for each transaction.) TPC-C was specifically designed to address many of the shortcomings of TPC-A. It does this in many areas. It exercises a much broader cross-section of database functionality than TPC-A. Also, the implementation rules are much stricter in critical areas such as database transparency and transaction isolation. Overall, TPC-C results will be a much better indicator of RDBMS and OLTP system performance than previous TPC benchmarks.

- *TPC-D*

TPC-D suite is a Decision-Support benchmark (OLTP env.) suite. It performs the Database stress test, simulation of a large database, and complex queries tests.

2.6.2 Hartstone

Hartstone is a benchmark for measuring various aspects of real time systems. This benchmark was designed at the Software Engineering Institute of Carnegie Mellon University.

2.6.3 PCBench/WinBench/NetBench/MacBench/ServerBench

PC Bench 9.0, WinBench 95 Version 1.0, Winstone 95 Version 1.0, MacBench 2.0, NetBench 3.01, and ServerBench 2.0 [FAQ] are the current names and versions of the benchmarks available from the Ziff-Davis Benchmark Operation (ZDBOp). These benchmarks are used to measure the overall prformance of DOS PC, WIN PC, Macintosh, etc.

2.7 Summary

Table 2.7 presents a summary of different benchmarks. An attempt is made to categorize the popular benchmarks on the basis of several characteristics. The characteristics are synthetic vs application, type of workload (ineger arithmetic, floating point arithmetic, scientific computing, supercomputing, disk I/O, NFS etc.). The table also includes the answers to several questions such as, whether or not the benchmark is CPU architecture sensitive, compiler sensitive, optimization sensitive, instruction cache sensitive, and, data cache sensitive.

We conclude from the study of popular benchmarks that generic benchmark rating is like mileage rating of automobiles. These ratings guarantee that the product will never exceed the quoted performance. As automobile vendors are required to include the following line with the performance rating: "Your actual mileage may vary according to road conditions and driving habits", something similar should be included with the benchmark ratings of computer systems.

The benchmarking programs should run under real life load for a long time duration. The different instances of execution time (or some rating) for the same system load can be averaged. This helps users to get a measure of system performance for varying system load. The next chapter presents the design of the performance evaluation tool which measures the load on different components of a system, runs several benchmarking programs designed to generate different types of workload, and presents the measure of system performance with the current system load.

Benchmark Name	Type	Workload	AS	CS	OS	IChS	DCh
Whetstone	Synth.	Floating point	N	Y	Y	N	N
Dhrystone	Synth.	System programming	Y	Y	N	Y	Y
Digital Review	Synth.	Integer and Floating point	N	N	Y	Y	Y
Sim	Appli.	Integer	Y	N	N	N	Y
Fhourstones	Appli.	Integer	N	Y	Y	N	N
HeapSort	Appli.	Integer	Y	Y	Y	N	Y
Hanoi	Appli.	Integer	Y	N	N	N	N
Linpack	Synth.	Floating point	N	Y	Y	N	Y
Livermore Fortran Kernal	Appli.	Floating point	N	Y	Y	Y	Y
NAS Kernel Benchmark	Appli.	Supercomputing	Y	Y	Y	Y	Y
EDN Benchmarks	Appli.	A mix of CPU bound instr.	N	Y	Y	Y	Y
Stanford Int. and Float.	Synth.	Integer and Floating point	Y	N	Y	Y	Y
Los Alamos Benchmarks	Appli.	Supercomputing	Y	Y	Y	Y	Y
Doduc	Appli.	Scientific computing	Y	Y	Y	N	Y
Euroben Benchmarks	Synth.	Scientific (multiprossors)	Y	Y	Y	N	Y
Perfect Benchmark	Appli.	Supercomputing	Y	Y	Y	N	Y
Flops	Synth.	Scientific computing	Y	Y	N	N	N
Stream	Synth.	Scientific computing	Y	Y	N	N	Y
IOBENCH	Synth.	Disk I/O					
IOZONE	Synth.	Disk I/O					
NFS Stone	Synth.	NFS I/O					
NHFS Stone	Synth.	NFS I/O					
Netperf	Synth.	Network					
SPEC	Synth.	CPU Int/Float, Disk, NFS	N	N	Y		
Khornerstone	Synth.	CPU Int/Float, Disk I/O	Y	Y	Y		
AIM	Synth.	CPU,Memory,Disk,Database	Y	Y	Y		
SYSMARK	Synth.	DOS/WINDOWS Applications		Y	Y		
Byte Unix Benchmarks	Synth.	Unix sys calls, CPU, disk	N	Y	Y		
TPC	Synth.	Database and Tran. Process	N	Y	Y		

AS - CPU Architecture Sensitive (RISC/SISC), **CS** - Compiler Sensitive, **OS** - Optimization Sensitive, **IChS** - Instruction Cache Sensitive, **DChS** - Data Cache Sensitive.

Table 1: A summary of the benchmarks

Chapter 3

Design of the Performance Evaluation Tool

The Performance Evaluation Tool records the different components of current system load, runs a series of test programs, measures the time taken by each test and reports the results. This tool does not provide a single number to represent the system performance, as most of the other benchmarks do, but prints the time taken by different tests along with the components of system load which might affect the execution time of that test.

3.1 Scope of the Performance Evaluation Bench

The test suit is designed to measure the performance of the most utilized components of a system. The current scope of this tool is limited to measuring the performance of CPU, disk I/O, NFS, and network. The tool contains synthesized benchmark programs designed to approximate the workload of the IITK environment. However, this tool can be used in other environments by giving a different weight to the test programs and by including some more test programs in the test suite. The tool is designed to work on the Unix (or Unix clone) platforms but it can be ported to other platforms by modifying the timing routines and the programs that capture the system load.

3.2 Collecting System Parameters and Current Load

The different system parameters, that are significant in the comparison of two systems are CPU type and word size, number of processors (in multiprocessor systems), CPU clock frequency, cache size, caching policy, size of physical memory, number of hard disks, disk controller technology, DMA, type and bus width of network adapter, maximum speed of underlying network, and name and release of operating system. Unfortunately not all of these parameters can be obtained automatically. Some systems store the values of a few parameters in header files such as number of clock ticks in a second, and some other parameters can be obtained through the system call interface. For example, OS name, release, and CPU name can be obtained by the **uname** system call. Linux provides some information about the CPU, devices, DMA, and memory through the **proc** file system[1]. Some parameters which are transparent to kernel, such as cache and network speed, can't be obtained using the OS services. The only possible way to get this information about the system parameters is to get it manually.

System load can be obtained by the services and interfaces provided by the operating system. Major sources for this information are the **proc** file system, **/dev/kmem** file[2], **ps** program, **uptime** program, and **w** program. Since reading the **proc** file system and **/dev/kmem** file requires root access on the system, the utility programs (ps, uptime, w, etc.) remain the only source to get information as a normal user.

Following attributes of system load are available:

- Number of active login sessions
- Load average (number of jobs in the run queue for the last 5 seconds, the last 30 seconds, and the last 60 seconds).
- Number of processes running on the system.

[1]proc is a pseudo-filesystem which is used as an interface to the kernel data structures. It is normally mounted on /proc directory

[2]/dev/kmem is a character device file that is an image of the kernel virtual memory. Byte addresses in kmem are equivalent to mapped kernel memory addresses.

- Detailed information about each process

Following information about each process is significant in measuring system load:

- **Flag**: This field tells if the process is in core, is being traced, is doing physical I/O, etc.
- **State**: Symbolic process state such as Sleeping, Idle, Zombie, Running or Ready.
- **WCHAN**: Address of the event on which a process is waiting. This can be used to know whether a process is waiting on disk I/O, or on Network I/O, or on tty input, or waiting for the termination of a child process.
- **Uid**: User id is used to check whether the process is a root or non root process. Root processes are normally daemon processes which run all the time.
- **VSIZE**: Process's virtual address size.
- **STIME**: Start time and date of process.
- **USERTIME**: CPU time used in user space.
- **SYSTIME**: CPU time used in system space.
- **Sleep Time**: Time passed by process in sleeping.
- **PCPU**: Percentage CPU usage.
- **PMEM**: Percentage real memory usage.
- **MAJFLT**: Number of major page faults.
- **MINFLT**: Number of minor page faults.
- **BLKIN**: Block input operations.
- **BLKOUT**: Block output operations.
- **IVCSW**: Involuntary context switches.

Not all systems provide all the attributes listed above.

The following structure definition is used to store the number of login sessions, load average, and *uptime* information.

```
struct uptime {
  int days;    /* Number of days,              */
  int hours;   /* hours,                       */
  int minutes; /* and minutes since            */
               /* since system was booted.     */
  int users;   /* No of users (login sessions) */
  float load1; /* load average (last  5 seconds)*/
  float load2; /* load average (last 30 seconds)*/
  float load3; /* load average (last 60 seconds)*/
};
```

The **uptime** program provides the values for all fields of the *uptime* structure. The output of utility programs and the interface of the **proc** file system are not identical on different platforms. This tool contains a collection of routines for each system that can be used to record the components of system load. Other routines such as those used for analysing system load and running test programs, are same for all the systems. A well defined interface is provided for passing information among the system specific and system independent routines. The following structure definition is used to store the attributes of a processes.

```
struct processes_info {
  int uid;         /* User id                  */
  char state[4];   /* Symbolic state           */
  int cputime;     /* time in seconds x 100    */
  int usertime;    /* time in seconds x 100    */
  int systime;     /* time in seconds x 100    */
  int starttime;   /* start time and date      */
  int flag;        /* Flag                     */
  int majflt;       /* No of major page faults     */
  int minflt;       /* No of minor page faults     */
  int inblock;      /* Block input operations      */
  int outblock;     /* Block output operations     */
  int pcpu;         /* 100 x %CPU                  */
  int pmem;         /* 100 x %MEM                  */
  int vsize;        /* Size of virtual address     */
  int rsize;        /* Real memory size            */
  char wchan[4];    /* Add. of the event for wait */
  char command[8];/* Command line                */
  int etime;        /* Elapsed time since started */
  struct processes_info *next;
};
```

System specific routines pass a linked list to the system independent routines. Above structure is the definition of one node of the linked list.

No single common interface is available to get information about processes. Following sections provide a brief overview of the techniques used on different systems to collect the process attributes.

3.2.1 DEC/OSF1 and Digital Unix

Kernel exposes process information through **proc** file system. For each running or zombie process, there is an entry in the system process table, which appears as a file name in the /**proc** directory. The file name is the decimal representation of the process id. The '**ioctl**' system call is used to get meaningful information from the file in the **proc** file system. However, the read permission for the files in **proc** file system is restricted only to the owner of the process. Therefore, only root process can read all the files.

'ps' program in DEC/OSF1 is a *suid* program and it provides all the information needed about the processes. The '*ps*' program takes field names to be listed, as command line parameters with -o option.

3.2.2 SunOS 5

SunOS provides the proc file system similar to that of DEC OSF/1, but the '**ps**' program is not as powerful as on DEC/OSF1. It lists only flag, state, uid, rsize, wchan, stime, cputime, and command fields of the fields that this tool uses.

3.2.3 SunOS (Solaris version)

The Solaris version of the SunOS has the same **proc** file system interface as SunOS 5. However, the '**ps**' program is more powerful than that of SunOS 5. Some more fields such as pcpu, pmem, vsize, and etime are listed. Only selected fields can be listed in the output of '**ps**' as is done in the case of DEC/OSF1.

3.2.4 HP-UX

HP-UX does not provide the **proc** file system. Process information is obtained from the **/dev/kmem** file using '**ioctl**' system call. '**ps**' program provides only flag, state, uid, rsize, wchan, stime, cputime, and command fields.

3.2.5 IRIX64

IRIX64 (on SG machines) provides the **proc** file system similar to DEC/OSF1. The '**ps**' program allowis **-o** option to specify the field names on comand line. Some fields such as '**majflt**', '**minflt**', '**inblock**', '**outblock**' etc. are not available in the output of '**ps**'.

3.2.6 Linux

Linux has an exhaustive **proc** filesystem. There is a subdirectory for each running process instead of a single file as in other systems. The subdirectory is named by the process ID. Each subdirectory contains the following pseudo-files and directories.

- **cmdline**: This holds the complete command line for the process, unless the whole process has been swapped out or it is a zombie.

- **cwd**: This is a link to the current working directory of the process.
- **environ**: This file contains the environment for the process.
- **exe**: A pointer to the binary which was executed. This appears as a symbolic link.
- **fd**: This is a subdirectory containing one entry for each open file. named by its file descriptor. It is a symbolic link to the actual file.
- **maps**: A file containing the currently mapped memory regions and their access permissions.
- **mem**: This is the not same as the /dev/mem device. despite the fact that it has the same device numbers. The **/dev/mem** device is the physical memory, while the mem file here is the virtual memory of the process.
- **root**: Root points to the file system root. set by the **chroot** system call.
- **stat**: Status information about the process. This is used by the '**ps**' program.

Some other files are present in **/proc** directory which are not specific to a process but are global to the system. These files are:

- **cpuinfo**: This is a collection of CPU and system architecture dependent items.
- **devices**: Text listing of major numbers and device groups.
- **dma**: This is a list of the registered ISA DMA (direct memory access) channels in use.
- **filesystems**: A text listing of the filesystems which are supported by the kernel. This is used by '**mount**' program to cycle through different filesystems when none is specified.
- **interrupts**: This is used to record the number of interrupts per IRQ on the i386 architecture.

- **ioports**: This is a list of currently registered Input-Output port regions that are in use.
- **kcore**: This file represents the physical memory of the system and is stored in the core file format.
- **kmsg**: This file can be used instead of the **syslog** system call to read kernel messages.
- **ksyms**: This holds the kernel exported symbol definitions used by the **modules** tool to dynamically link and bind loadable modules.
- **net**: This is a subdirectory which contains various net pseudo-files. Each one gives the status information of a part of the networking subsystem.
- **pci**: This is a listing of all PCI devices found during kernel initialization and their configuration.
- **scsi**: A directory which contains a file for each SCSI host in the system.
- **self**: This directory refers to the process accessing the /**proc** filesystem, and is identical to the numerically named subdirectory for the process.
- **stat**: Kernel/system statistics. This file contains the statistics of cpu, disk, pages, swaps, interrupts, context switches, and boot time.
- **uptime**: This file contains two numbers: the uptime of the system (seconds), and the amount of time spent in idle process (seconds).
- **version**: This file contains the string which identifies the kernel version that is currently running.

3.3 Measuring the Network Load

Network load relates to the whole LAN and not to a particular system. One server runs on each LAN to measure the network traffic on it. The client program connects to the server (IP address of the machine and TCP port of the server are known to the client) and gets the total number of packets and the average packet size during the last 'n' minutes. Server uses the **pcap** (packet capture) library. Authors of this library are Van Jacobson, Craig Leres and Steven McCanne, of the Lawrence Berkeley National Laboratory, UC Berkeley. This library provides a high level interface to packet capturing systems. All packets on the network, even those destined for other hosts, are accessible through this mechanism. The server runs with root privilege and the network interface is put into promiscuous mode. Design of a network monitoring system for IITK network is discussed in [Rao97].

Parameters	DEC/OSF1	SUN5	Solaris	IRIX	HPUX	Linux
CPU type	uname	uname	uname	uname	uname	proc
Cache size	manual	manual	manual	manual	manual	manual
RAM size	manual	manual	manual	manual	manual	proc
Disk Controller	manual	manual	manual	manual	manual	proc
Network Adapter	manual	manual	manual	manual	manual	proc
OS type	uname	uname	uname	uname	uname	uname
Losd average	uptime	uptime	uptime	uptime	uptime	proc
No. of logins	uptime	uptime	uptime	uptime	uptime	proc
No. of processes	ps, proc	ps, proc	ps, proc	ps, proc	ps, kmem^R	ps, proc
CPU time	ps, proc^R	ps, proc^R	ps, proc^R	ps, proc^R	ps, kmem^R	ps, proc
Page faults	ps, proc^R	proc^R	proc^R	proc^R	kmem^R	proc
States	ps, proc^R	ps, proc^R	ps, proc^R	ps, proc^R	ps, kmem^R	ps, proc
Real Memory Usage	ps, proc^R	ps, proc^R	ps, proc^R	proc^R	kmem	proc
Virtual Memory Usage	ps, proc^R	proc^R	ps, proc^R	*ps*, proc^R	kmem	proc
Network Statistics						proc
Network load	libpcap	libpcap^R	libpcap^R	libpcap^R	libpcap^R	libpcap^R

^{R} root priviledge required

Table 2: Accessing Parameters in Different Systems

3.4 Measuring the System Performance

This tool includes several test programs designed to synthesize the workload of IITK computing environment. As discussed in the first chapter, the workload of IITK is a mix of interactive login sessions and batch jobs. Disk access is a mix of local disk I/O (most of the users prefer to work on the system hosting their home directory), and NFS I/O (Working on workstations requires access to the user's files on the server's disk). On the higher level, the network load consists of the TCP load due to the remote login sessions, X-traffic, RPC (mainly due to NFS), and HTTP. File transfer using **ftp** is not very frequent but it takes considerable bandwidth.

Following sections present an overview of the test programs used by this tool.

3.4.1 Integer Only Test

Not found in real life applications but 'integer only' performance of a system directly measures the power of its CPU in integer arithmetic. This test includes a mix of addition, subtraction, multiplication, division, assignment, increment, and conditional jump instructions. A function containing arithmetic operations is called in a loop. The loop runs a fixed number of times.

3.4.2 Floating Point Arithmetic Tests

This test is not actually a pure floating point test but a mix of several numerical operations. It includes the arithmetic operations on array elements, conditional jumps, integer arithmetic, Trigonometric functions (atan, sin, cos) and standard functions (sqrt, exp, log). The idea of various floating point operations is taken from Whetstone.

3.4.3 Fast Fourier Transform

This suite includes the routines to compute Hartley transform, Fourier transform, inverse Fourier transform, real-valued Fourier transform and inverse of the real-valued Fourier transform of 'n' points. This suite is a good representative for the CPU bound jobs with mixed type of instructions. The code is written by Ron Mayer of Acuson.

3.4.4 Matrix Multiplication

Matrix multiplication is a common operation in several scientific programs in Engineering, Mathematics, and the Science. This program performs the matrix multiplication of large size matrices. The program is sensitive to the paging scheme used in virtual memory system and page size. Also, the distance of the first matrix element from the page boundary is critical for the performance.

3.4.5 N-Queens Problem

This program finds all the possible ways for 'N' queens to be placed on an NxN chessboard such that they do not capture one another. That is, so that no rank, file or diagonal is occupied by more than one queen. This program is a good example of recursion. The program synthesizes a typical scientific program that makes use of pointers, complex data structures, and recursion.

3.4.6 Unix System Calls Overhead Test

This suite includes various routines to synthesize the general Unix system call workload. The routines use fork, exec, pipe, dup, open, close, and some other system calls. Some short life duration processes (such as 'ls') are created periodically.

3.4.7 Compiler and linker's Performance Test

A 'C' program is compiled and linked with the same optimization options to compare the performance of C-preprocessor, C-compiler, and linker on different systems.

3.4.8 File System Performance Check

This suite includes the routines to check the file system's performance. The routines perform a number of operations such as read, write, lseek, sync, copy etc. using the file system services.

3.4.9 Disk I/O Performance Test

It performs a series of tests on a file of known size. The default size is 50 MB. A large file size is chosen to overwhelm the buffer cache. The idea is to make sure that these are real transfers to/from user space to the physical disk. The tests are:

▮ *Sequential Output*

- **Per character:** The file is written using the putc( ) stdio macro. The loop that does the writing is small enough to fit into any reasonable cache. The CPU overhead here is that required to do the stdio code plus the OS file space allocation.

- **Block:** The file is overwritten using the write system call. The CPU overhead should be just the OS file space allocation.

- **Rewrite:** Each block of the file is read with the read system call, dirtied, and rewritten with write system call. Since no space allocation is done, and the I/O is well-localized, this should test the effectiveness of the filesystem cache and the speed of data transfer.

▮ *Sequential Input*

- **Per-Character:** The file is read using the getc() stdio macro. Once again, the inner loop is small. This should check the performance of only stdio and sequential input operations.

- **Block:** The file is read using read system call. This should be a very pure test of sequential input performance.

▮ *Random Seeks*

This test runs several processes in parallel, doing a total of 4000 lseeks to random locations in the file. In each case, the block is read with read system call. In 10%

of cases, it is dirtied and written back with write system call. The idea behind the parallel seeking processes is to make sure that there is always a queued up seek.

3.4.10 NFS Performance Test

The following operations are performed on a NFS mounted directory [Sax93]:

- **Makedir**: Constructs subtree consisting of 15 directories. The directories are constructed hierarchically as well as in flat structure.
- **Remove dir**: Removes the subtree created in Makedir.
- **Copy**: Copies all the files from one directory to another.
- **Move**: Moves all the files from one directory to another.
- **Delete Dir**: Deletes all the files from the specified directory.
- **Create files**: Creates 100 files in a directory.
- **Read file**: Repeatedly reads a fixed number of bytes from a file.
- **Write file**: Repeatedly writes a fixed number of bytes into a file.

3.4.11 NFS Performance Test using NFS Stone

The popular benchmark program 'NFS Stone' discussed in section 2.3.3 is used to measure the number of *NFSStones* a NFS server can serve in a second.

3.4.12 Network Performance Tests Using Netperf

Netperf, a benchmark program form HP (discussed in section 2.4.2), is used to measure the TCP stream performance, TCP request/response performance, UDP stream performance, and UDP request/response performance.

Chapter 4

Results and Comparisons

4.1 Introduction

One of the important steps in every performance evaluation study is the presentation of final results. The ultimate aim of every performance analysis is to help in decision making.

Graphic charts are used to present performance results. There are a number of reasons why a graphic chart may be used for data presentation in place of textual explanation [Jai91]. A graphic chart saves a reader's time and presents the same information more concisely.

This tool runs on different machines and records the system load and test results over a long period of time. The tool includes a program to generate artificial load to measure the performance of an idle system under different load conditions.

The tests were run on a myriad of systems available at IIT Kanpur. The systems were:

- **agni**, a HP-9000/735 machine with 144MB RAM and SCSI hard disks running HP-UX release A.09.03 version C. Cost of the system at the time of purchase (in Sep 1994) was US$ 50,607.

- **pc47**, a Pentium 166Hz machine with 32MB RAM and an IDE disk running Linux 2.0.0. Cost of the system at the time of purchase (in Jan 1997) was Rs. 55,000.

- **sg2**, a 180 MHZ, Origin 200 Silicon Graphics (IP27 CPU board) machine with 128MB RAM and SCSI fast wide disks running IRIX64. Cost of the system at the time of purchase (in Feb 1997) was US$ 75,000.

- **cd3**, a 233MHZ DEC Alpha 2000 dual processor machine with 128MB RAM and fast wide hot swappable SCSI disks running OSF1 V3.2. Cost of the system at the time of purchase (in July 1996) was Rs. 1,200,000.

- **shakti**, is a 233MHZ DEC Alpha 2000 machine with 128MB RAM and SCSI disks running OSF1 V3.2.

- **cu1**, a 167MHZ Sun Sparc Ultra 1 machine with 128MB RAM and fast wide SCSI disks running SUNOS 5.5.1 (Solaris version). Cost of the system at the time of purchase (in June 1996) was Rs. 850,000.

For each of the above systems, the execution time (for tests) has been plotted against total no. of processes and no. of login sessions. This particular choice has been made because execution time and no. of processes/login sessions are good representatives of the system performance and system load, respectively.

4.2 Observations

Following are some of the inferences which can be made directly from the plots.

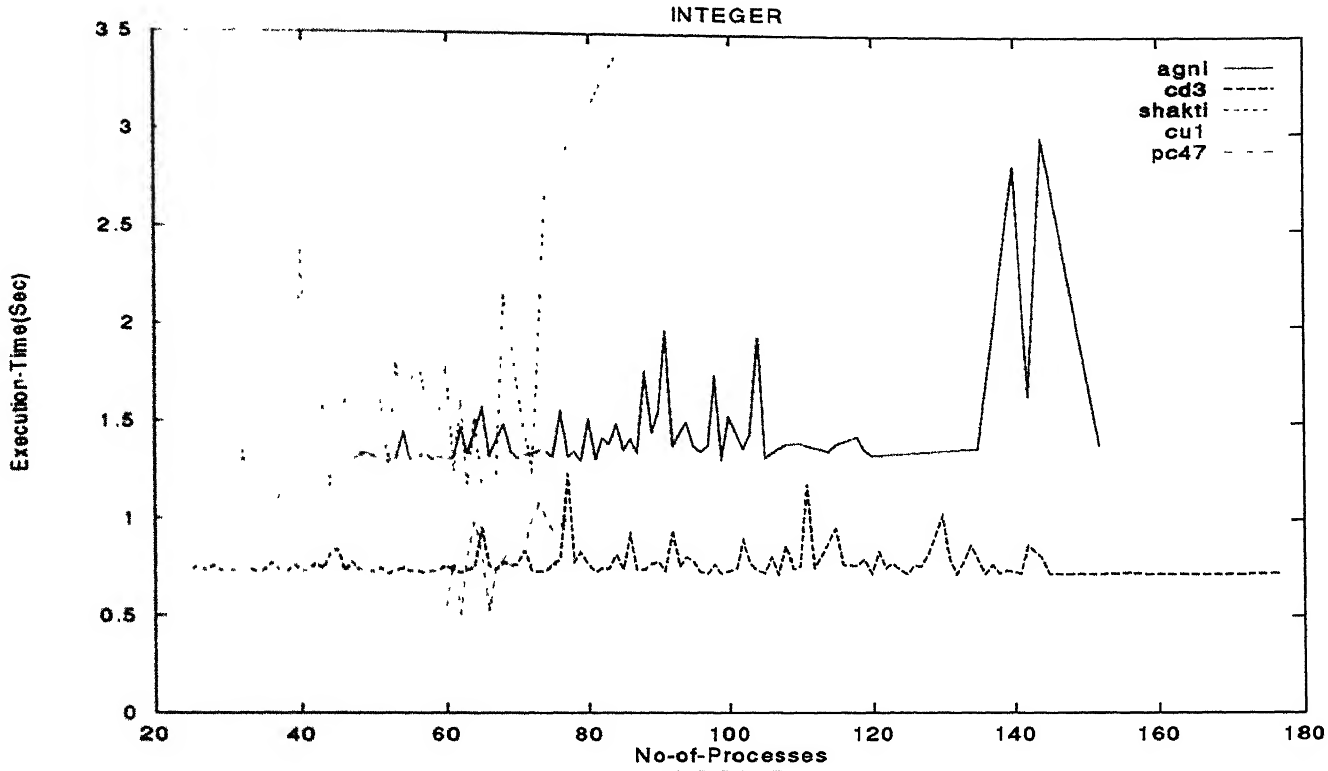
4.2.1 Integer Arithmetic Test

This test includes a mix of addition, subtraction, multiplication, division, assignment, increment, and conditional jump instructions. This test directly measures the integer arithmetic performance of the CPU since there is no system call and I/O overhead and no support from floating point coprocessor is required. Figure 2 shows the results. We gather following information from the graph.

- sg2 has minimum execution time for this test.

- cd3 gives a steady performance with increasing system load.

- shakti become very slow when the system load increases
- cu1 gives same performance as cd3.
- agni is slower than other systems (but shakti) and its performance goes down with increasing system load.
- pc47's performance goes down as the load increases but at low load it is better than all systems but sg2.



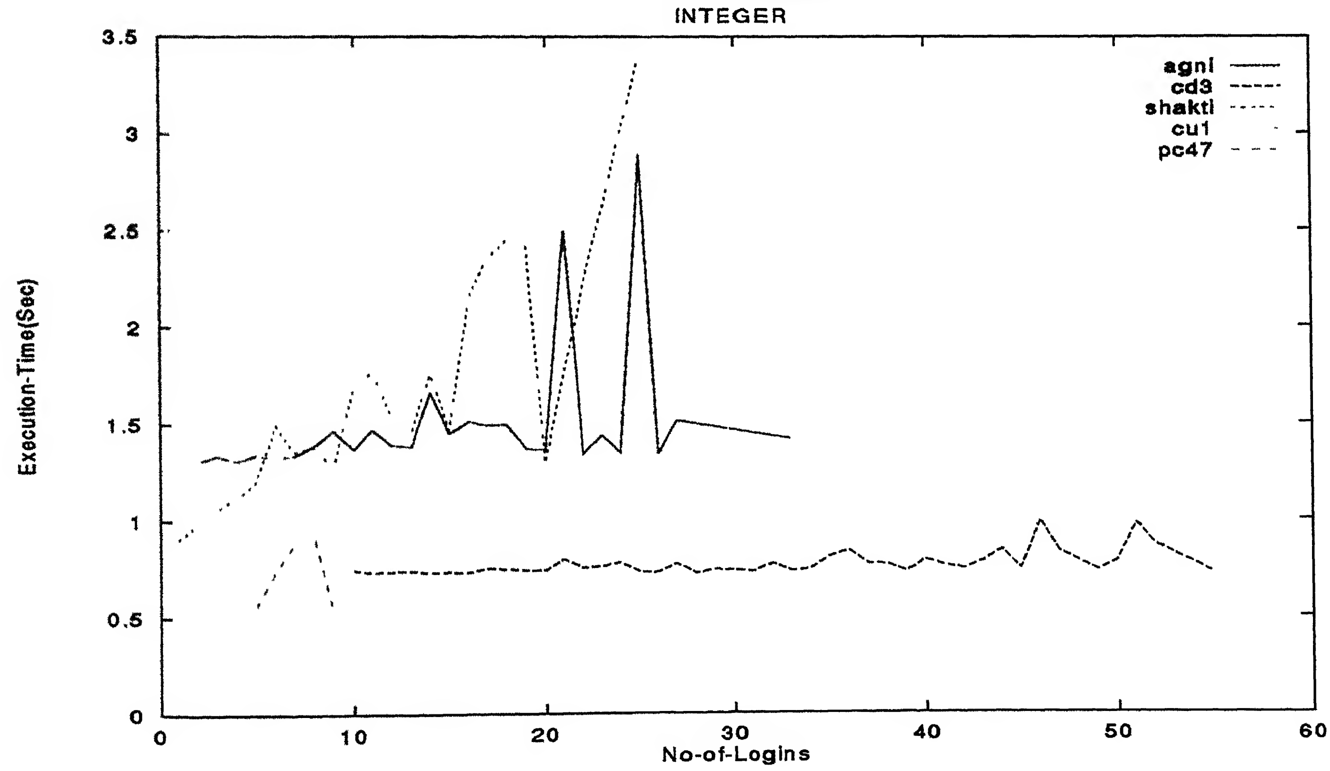
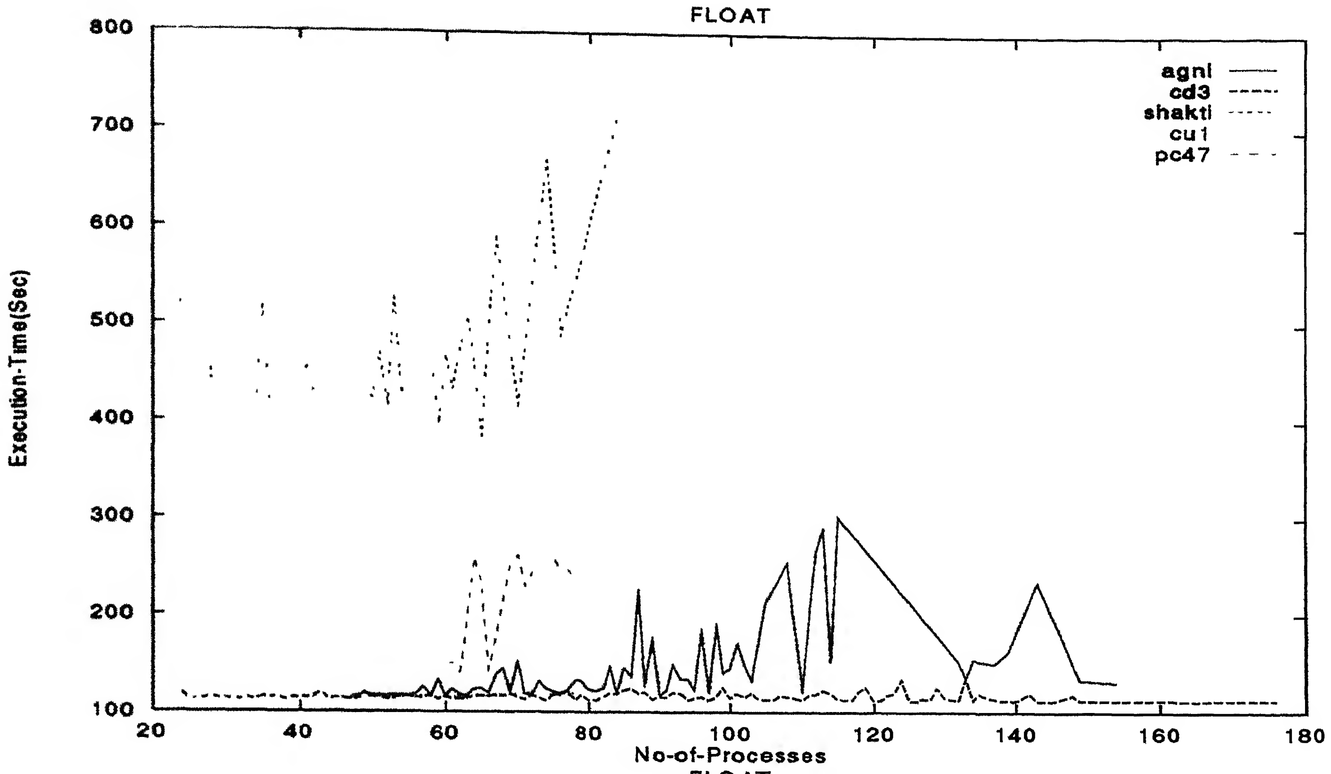


Figure 2: Integer Test

4.2.2 Floating-Point Arithmetic Test

This test is not actually a pure floating point test but a mix of several numerical operations. It includes the arithmetic operations on array elements, conditional jumps, integer arithmetic, Trigonometric functions (atan, sin, cos) and standard functions (sqrt, exp, log). Figure 3 shows the results of floating-point test. Following is a summary of observations from the graph.

- sg2 gives the best performance.
- cd3 is exceptionally stable with increasing system loads.
- Performance of shakti is very poor as compared with other systems.
- agni gives same performance as cd3 at low system load but it become slower with increasing system load.
- pc47 is slow than all other systems but shakti.
- cu1 is as good as cd3 but slower than sg2.



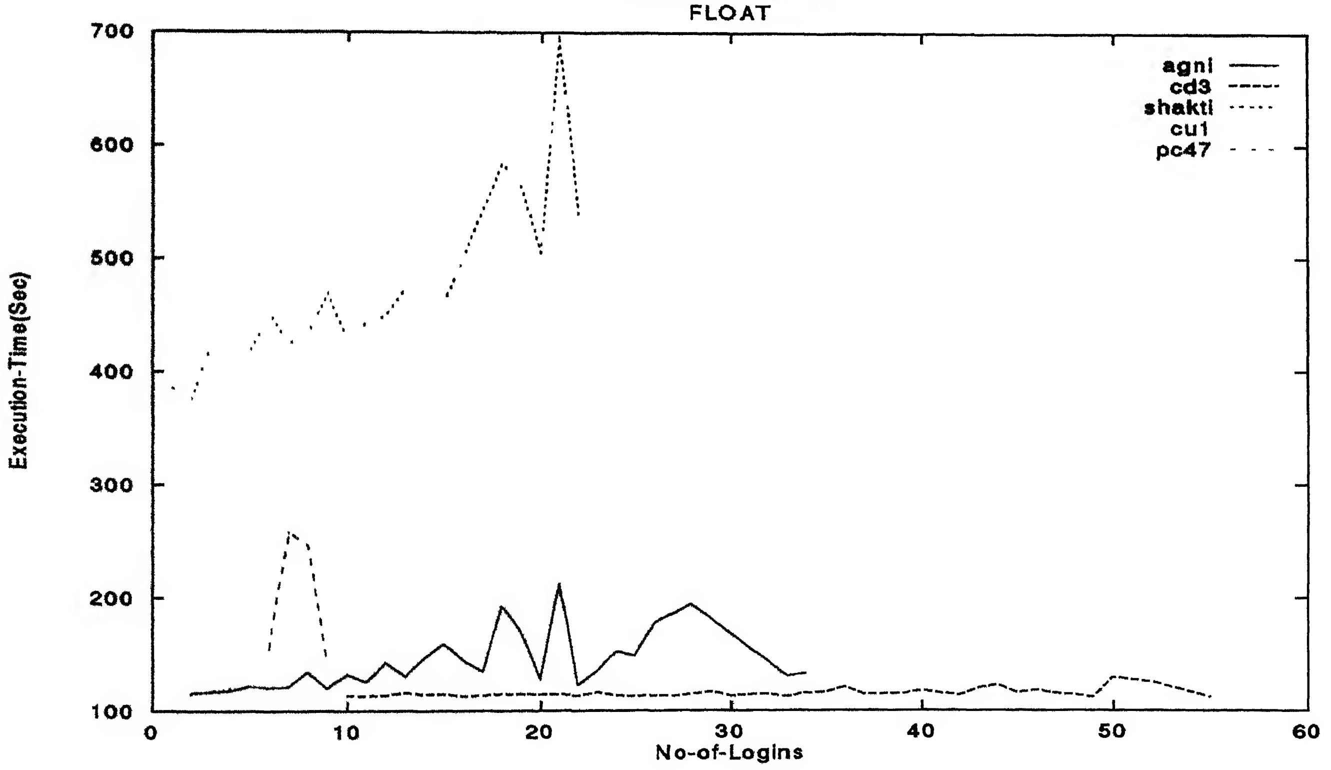
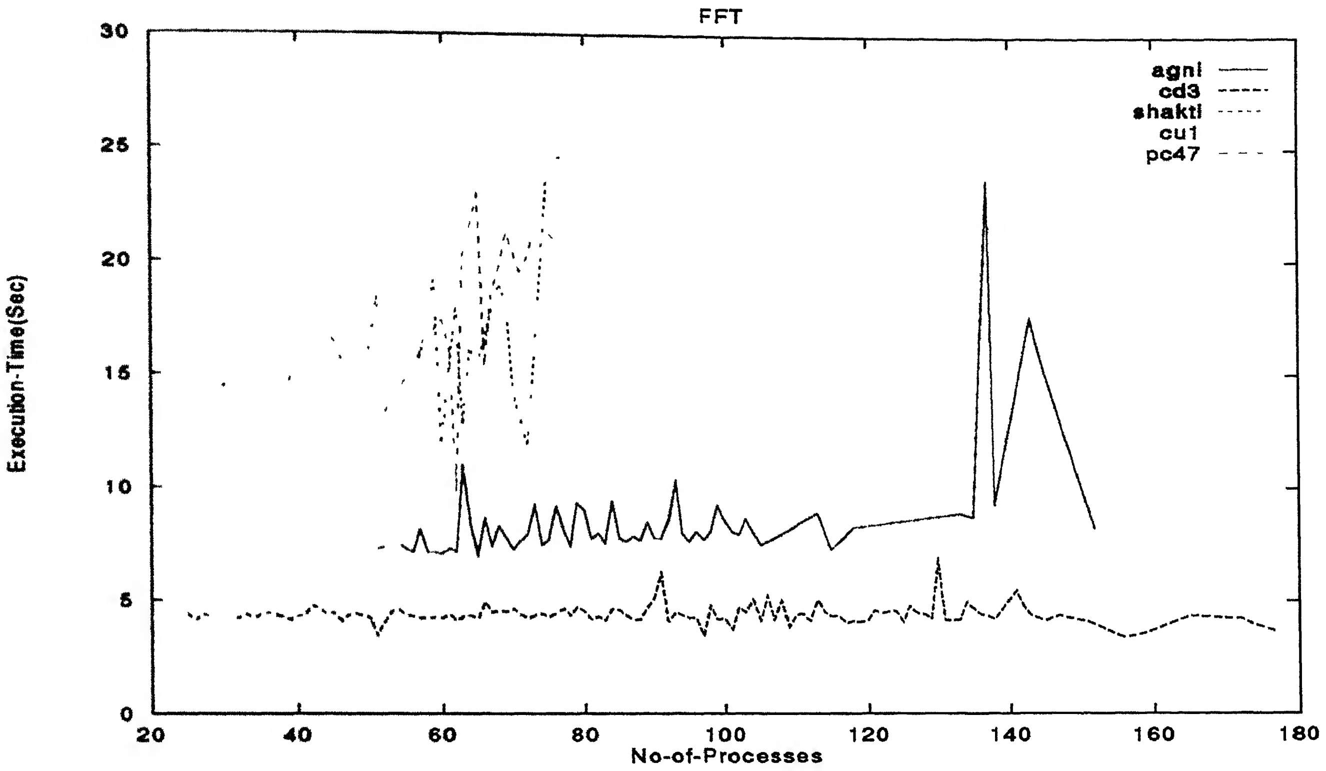


Figure 3: Floating Point Test

4.2.3 FFT Test

This suite includes the routines to compute Hartley transform, Fourier transform, inverse Fourier transform, real-valued Fourier transform and inverse of the real-valued Fourier transform. This suite is a good representative for the CPU bound jobs with mixed type of instructions. The graph is shown in figure 4. We notice following points from the graph.

- sg2 is again the best system.
- cd3 is slower than sg2 but shows stable performance at very high system load.
- cu1 gives same performance as cd3.
- agni is average but the performance goes low at higher system load.
- shakti is slow and become more slow at high load.
- pc47 gives similar performance as of shakti.



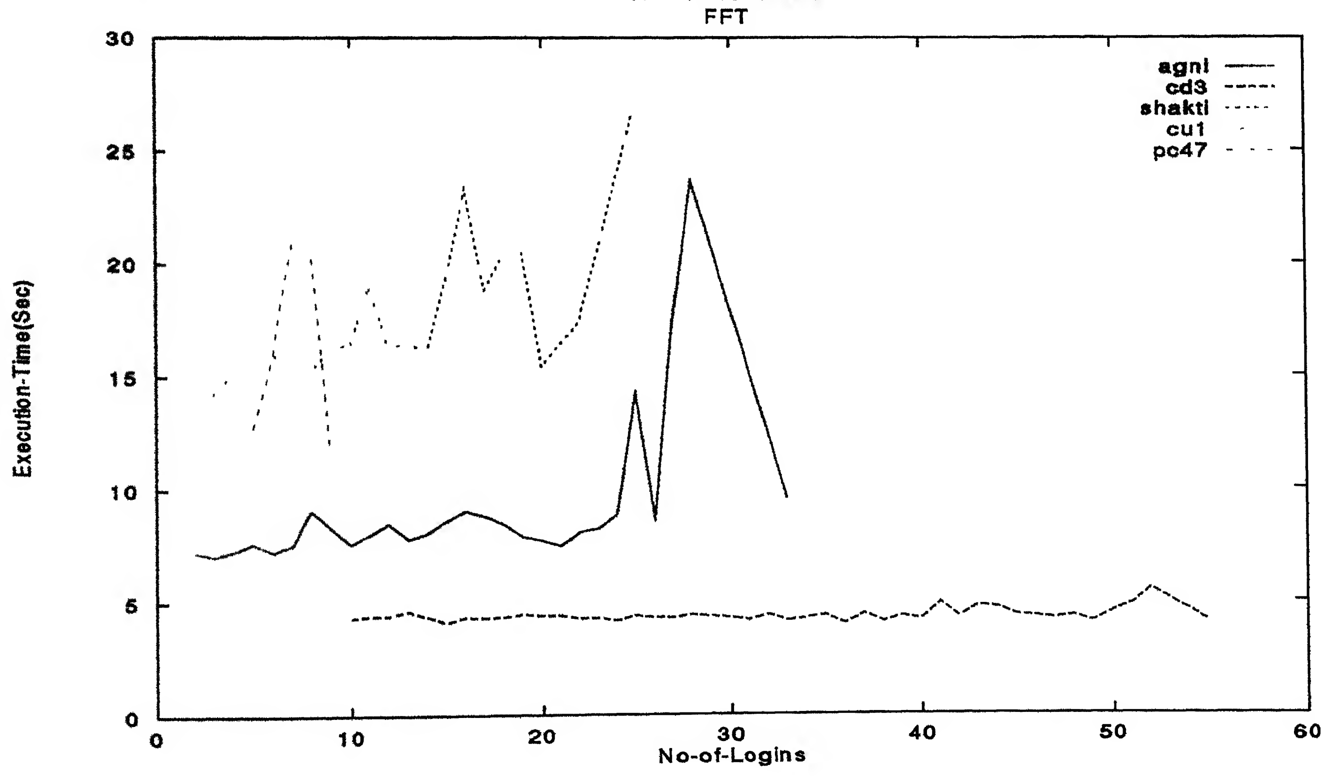
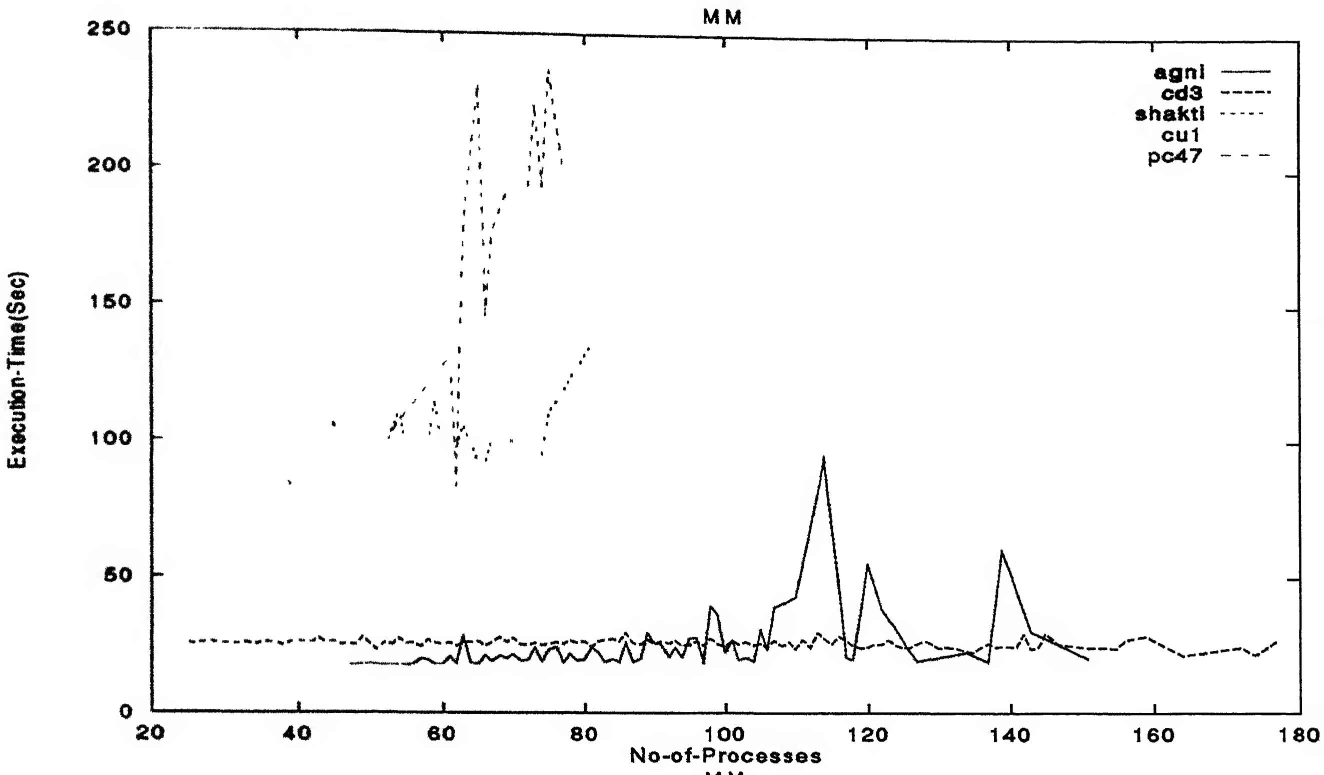


Figure 4: Fast Foriur Transform Test

4.2.4 Matrix Multiplication Test

This program performs the matrix multiplication of large size matrices. The test runs best on multiprocessor systems but the operation is common in several mathematical problems that run on single or dual processor systems. The graph is shown in figure 5. Following is a brief summary of observations.

- As expected pc47 is not a good system for this type of computations.
- shakti is slower than all other systems but pc47.
- sg2 is again the best.
- agni is as good as sg2 at low system load which is surprising.
- cd3 is again very steady in performance with increasing system load.
- cu1's performance is average.



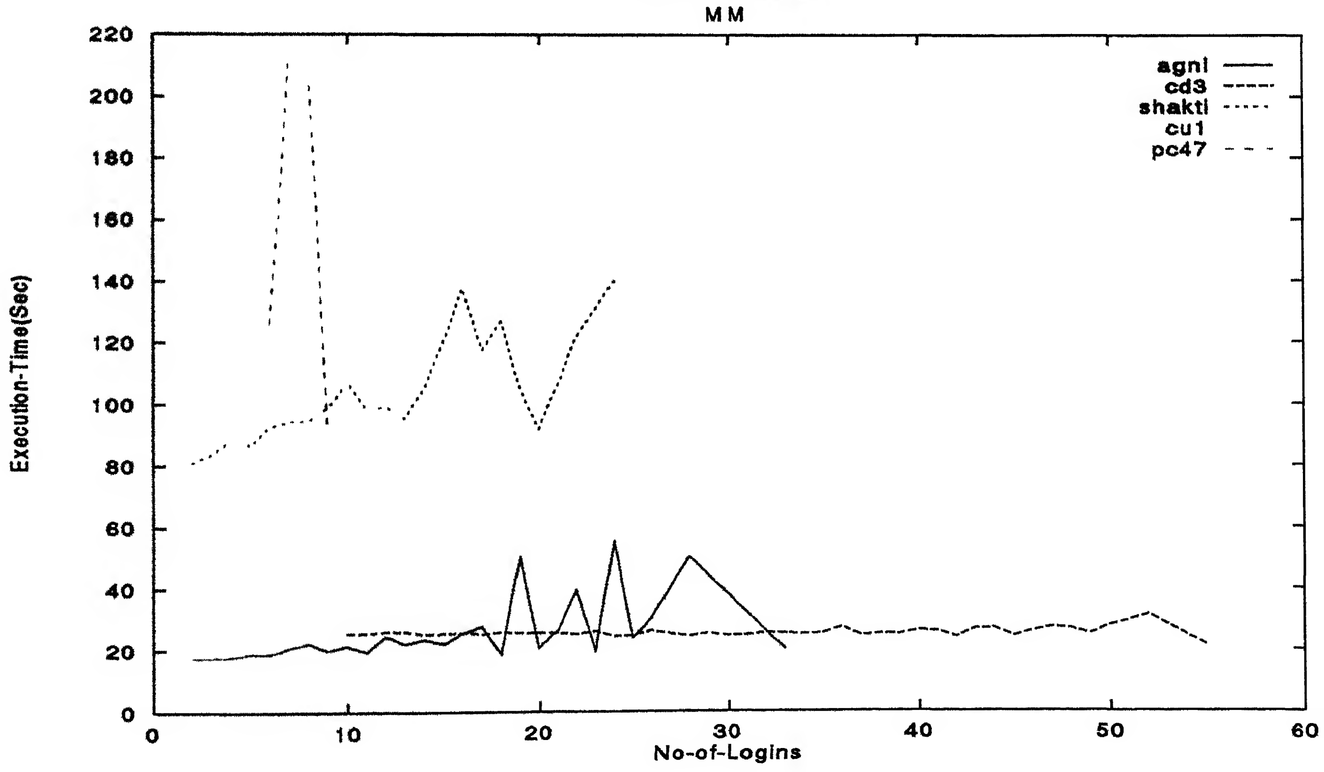
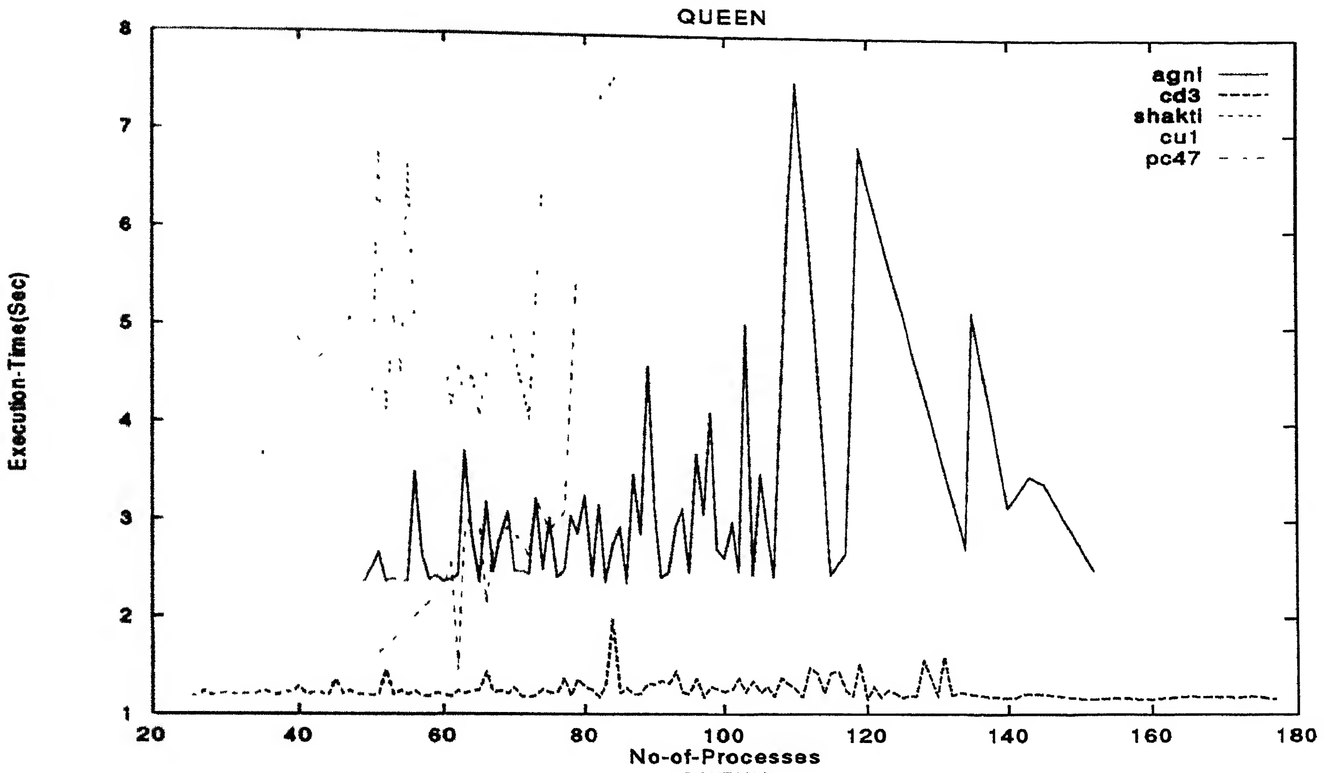


Figure 5: Matrix Multiplication Test

4.2.5 N-Queens Test

This program finds all the possible ways for 'N' queens to be placed on an NxN chessboard such that they do not capture one another. The program synthesizes a typical scientific program that makes use of pointers, complex data structures, and recursion. Only integer instructions are present in this program (but not much integer arithmetic). Figure 6 shows the graph. Following is a brief summary of observations.

- cd3 gives the best and steady performance for this test.
- cu1 is as good as cd3.
- sg2 is left behind by cu1 and cd3 in this test.
- pc47's performance varies in a large range. It becomes very slow at high load.
- agni is average but it shows fluctuating performance at higher load.
- shakti is again slower that all other systems and its performance goes very slow at higher load.



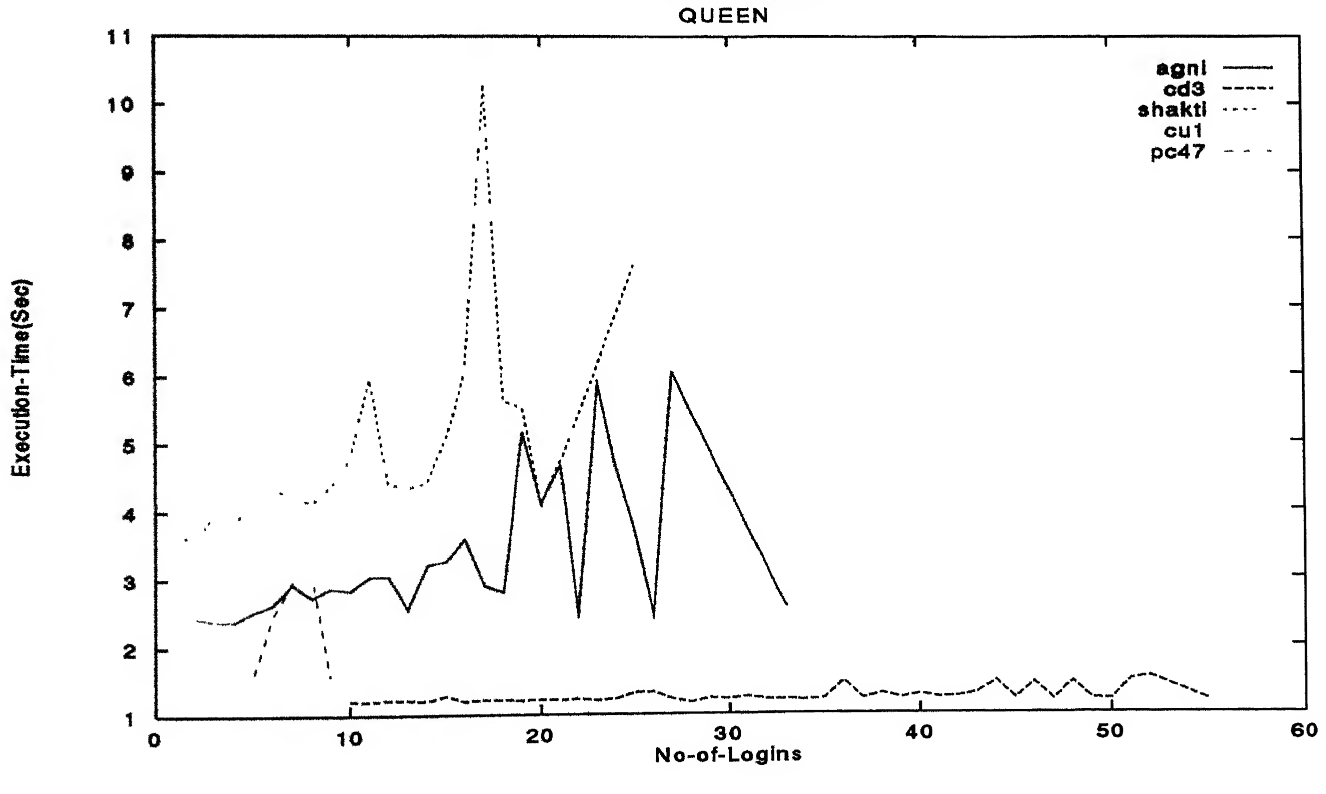
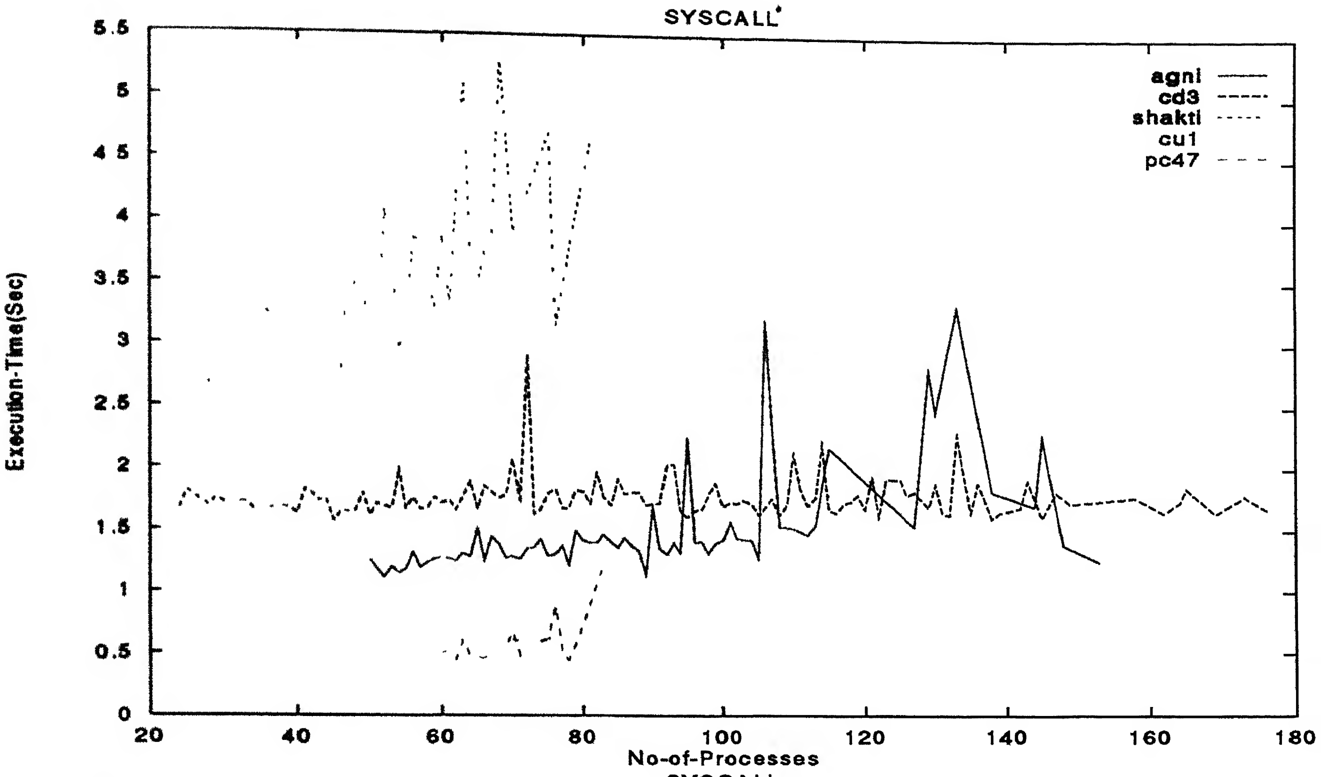


Figure 6: N-Queen Problem Test

4.2.6 Unix System Call Overhead Test

This suite includes various routines to synthesize the general Unix system call work-load. The routines use fork, exec, pipe, dup, open, close, and some other system calls. Some short life duration processes (such as 'ls') are created periodically. This test is intended to measure the system performance for normal interactive load. Figure 7 shows the graph. A summary of the observation presented below.

- Surprisingly pc47 gives the best performance in this test. Which means that Linux in the best operating system as far as system call overhead is concerned. But the performance goes down as system load increases. This may be because of the less powerful CPU on the PC as compared with other systems.
- cu1 is the second best system for this test. This was expected as the Solaris is considered as one of the best OS available.
- sg2 is slower than pc47 and cu1.
- agni is average at low system load but the performance decreases as the load increases.
- cd3 is slower than all other systems but shakti.
- shakti is the slowest and the performance goes down with increasing system load. We should note that both cd3 and shakti are DEC Alpha machine running OSF/1. The Unix emulation runs on the top of Mach OS. This may be the reason for low performance of these systems as compared with others in the lot.



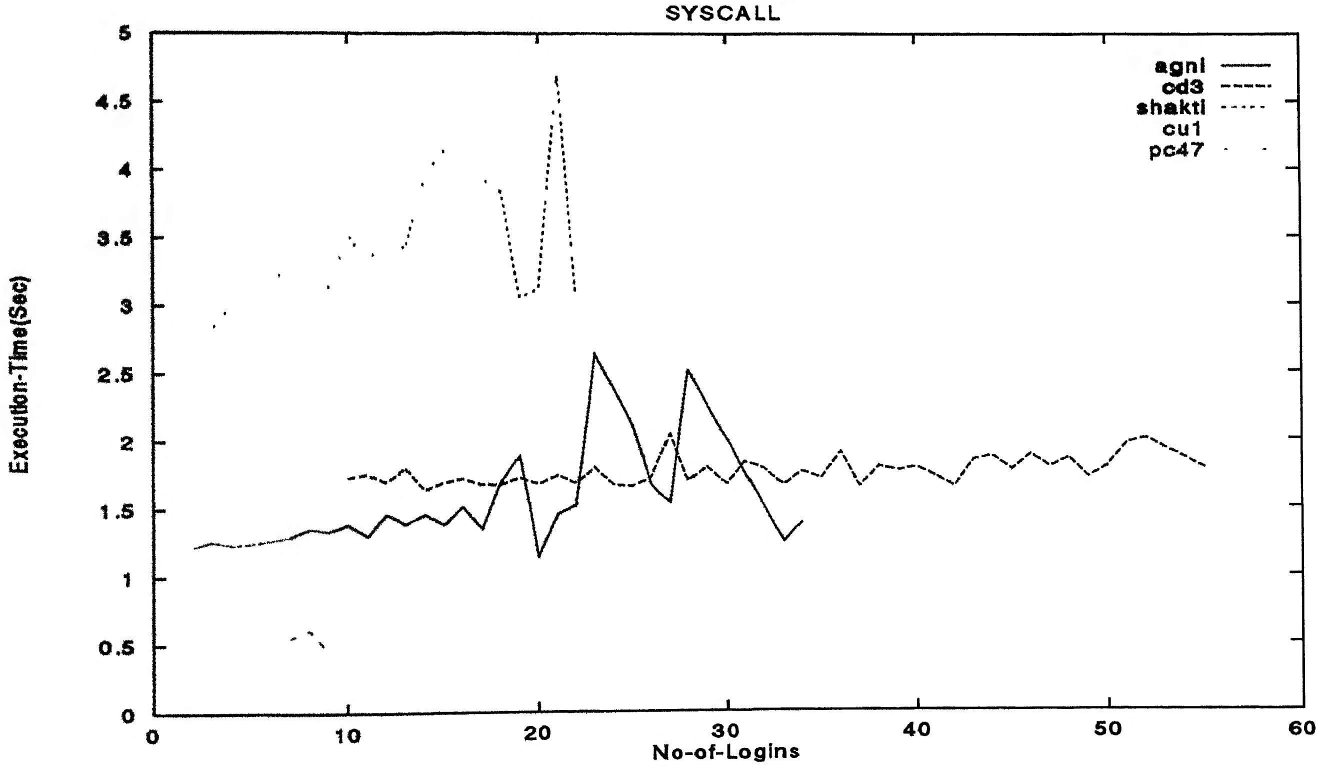
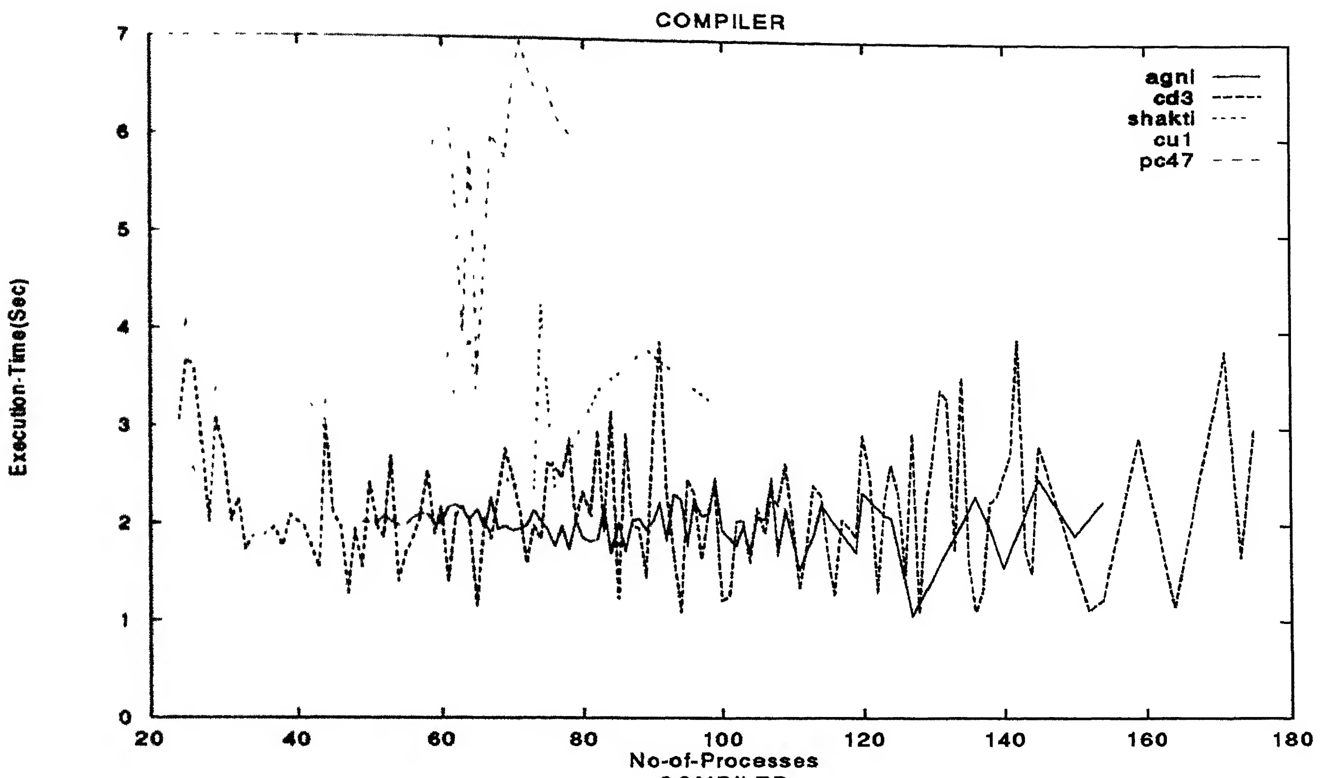


Figure 7: System Call Test

4.2.7 Compiler and linker's Performance Test

This test compiles and links a 'C' program, with the same optimization options to compare the performance of C-preprocessor, C-compiler, and linker on different systems. Figure 8 shows the graph. Observations are summarized below.

- sg2 gives the best performance at low system load but it goes slow when load increases.
- cd3 is a unstable in performance.
- The performance of cu1 is the best and it is unaffected by the increasing system load.
- agni is slower than cu1 and sg2 and it is some what stable at low system load, but shows fluctuating performance at higher load.
- pc47 is not a good machine for this job.
- shakti is slower than all other machines but pc47.



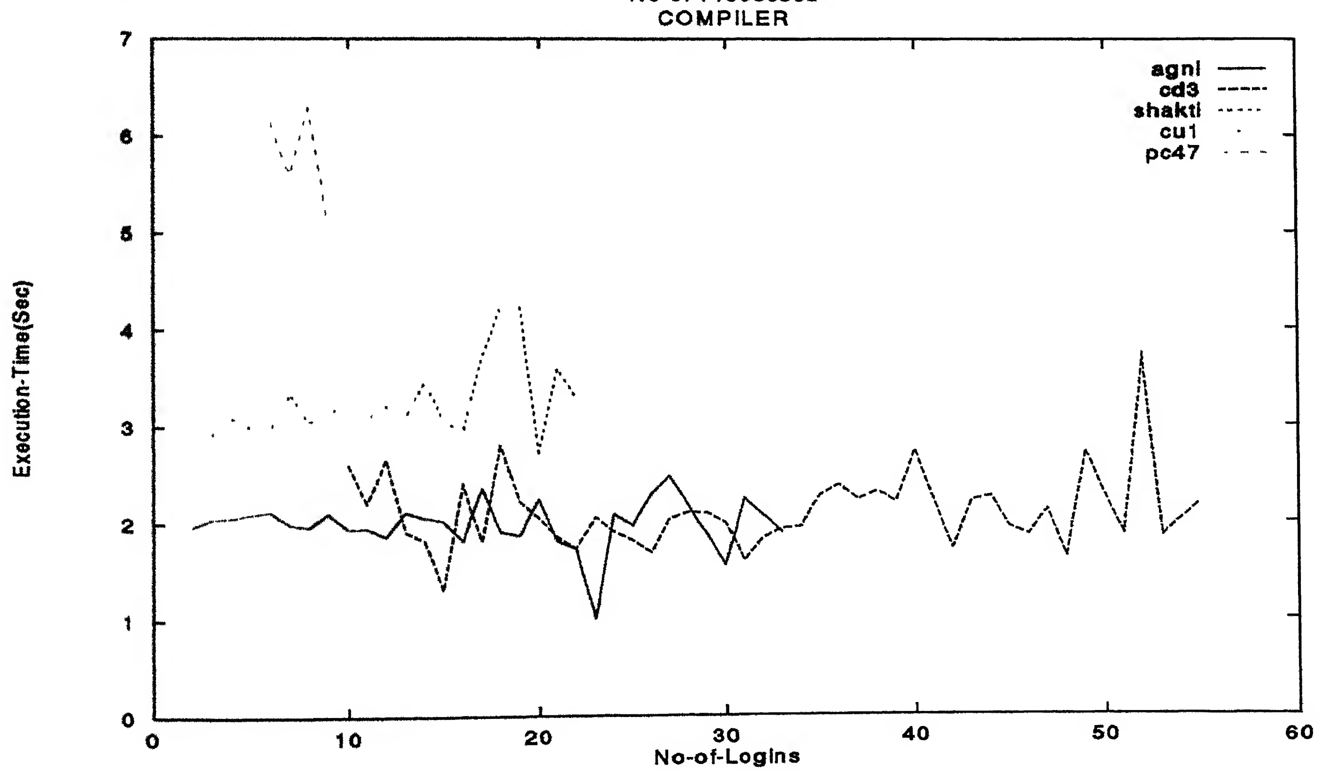
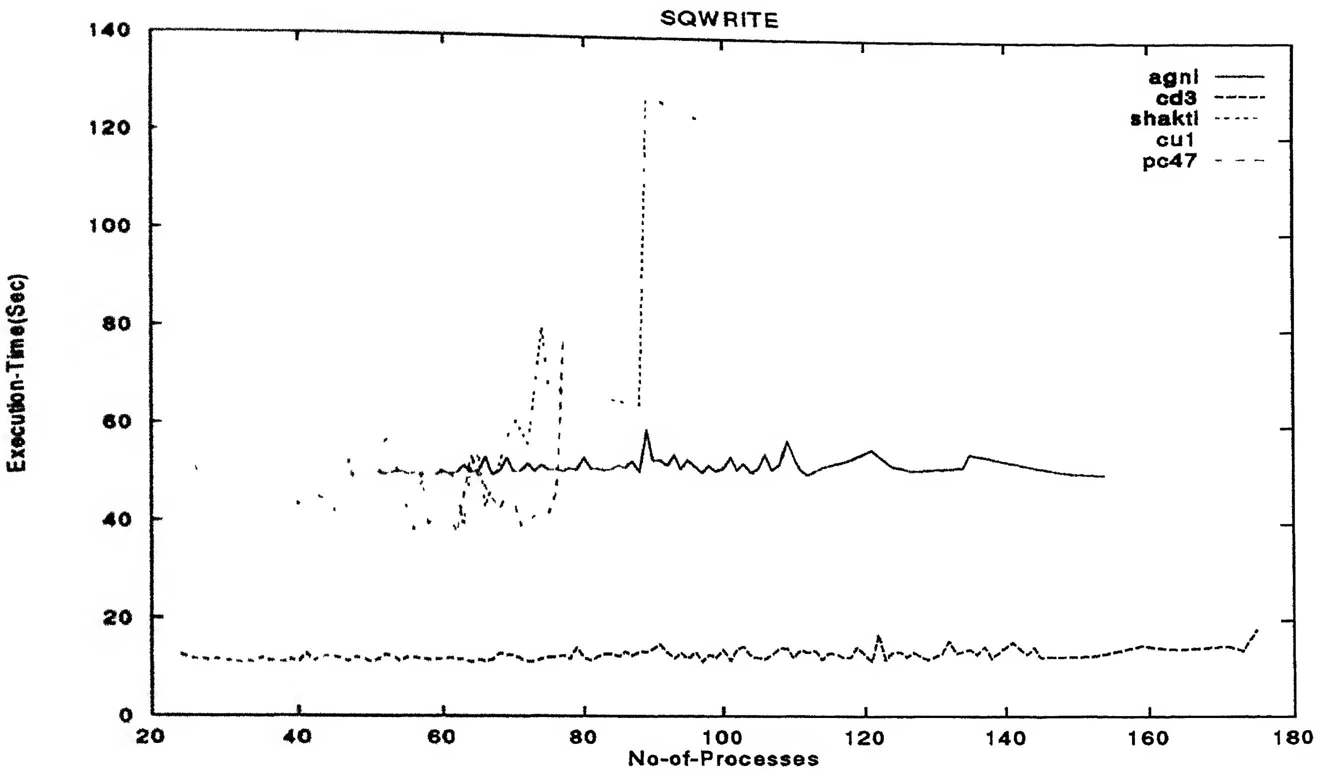


Figure 8: C Program Compilation Test

4.2.8 Sequential Write Test

In this test the file is written using the putc() stdio macro. The CPU overhead here is that required to do the stdio code plus the OS file space allocation. Figure 9 shows the graph. We observe following points from the graph.

- sg2 gives the fastest disk write time. The time remain constant with the increasing load on the system.
- cd3 is very close to sg2 and its performance is steady with the increasing load.
- cu1 is average and gives almost constant performance with increasing system load.
- agni is slower than sg2, cd3 and cu1 at high load and its performance remain steady with small fluctuation.
- shakti is close to agni at low system load but its become very slow at high load.
- pc47 is better than agni and shakti at low system load but its performance decreases at higher load.



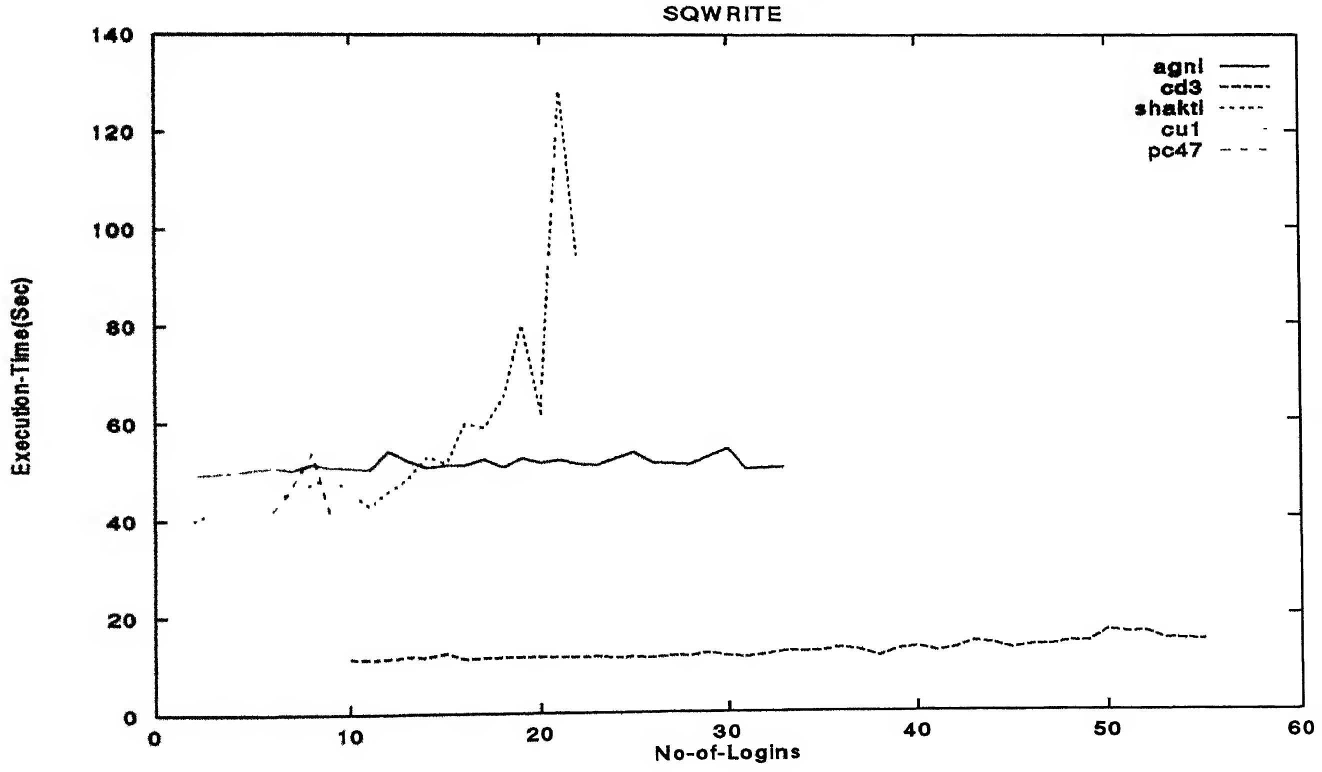
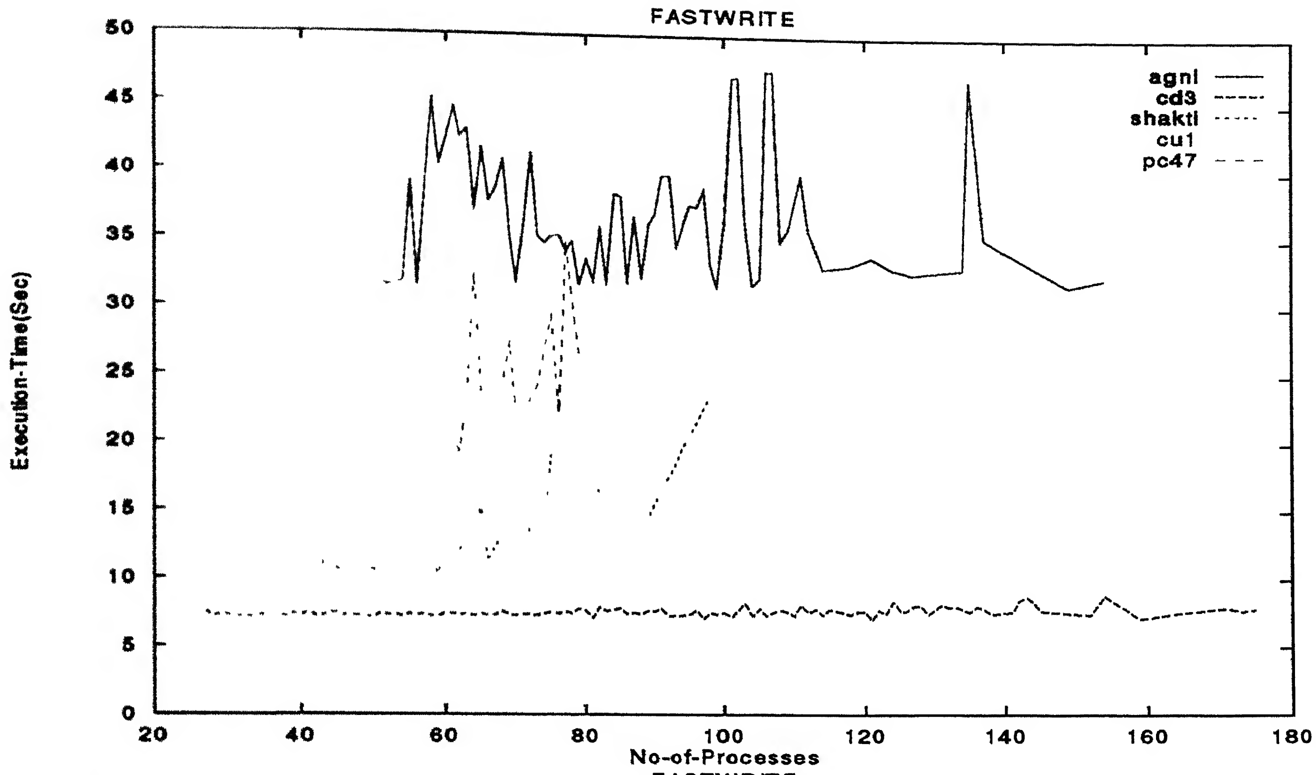


Figure 9: Sequential Write Test

4.2.9 Fast Write Test

The file is overwritten using the write system call. The CPU overhead should be just the OS file space allocation. Figure 10 presents the results as a graph. We gather following observation from the figure.

- sg2 is very fast in blockwise sequential write.
- cu1 gives the same performance as sg2 but its performance was not close to sg2 in per character sequential write. This means that disk-Io time for sg2 and cu1 is almost same and the more time taken by cu1 in per character write may be because of the slower CPU (CPU overhead is large in per char write), or the file system space allocation time is higher on cu1.
- cd3 improves in block write and its performance remain steady with higher load.
- agni gives a very poor performance as compared with other systems. We should note that the time taken by agni (around 40 seconds) in per block write is same as it took in per char sequential write.
- shakti also improves but its performance goes down at high load.
- The performance of pc47 is only better than agni and it deteriorates with increasing system loads.



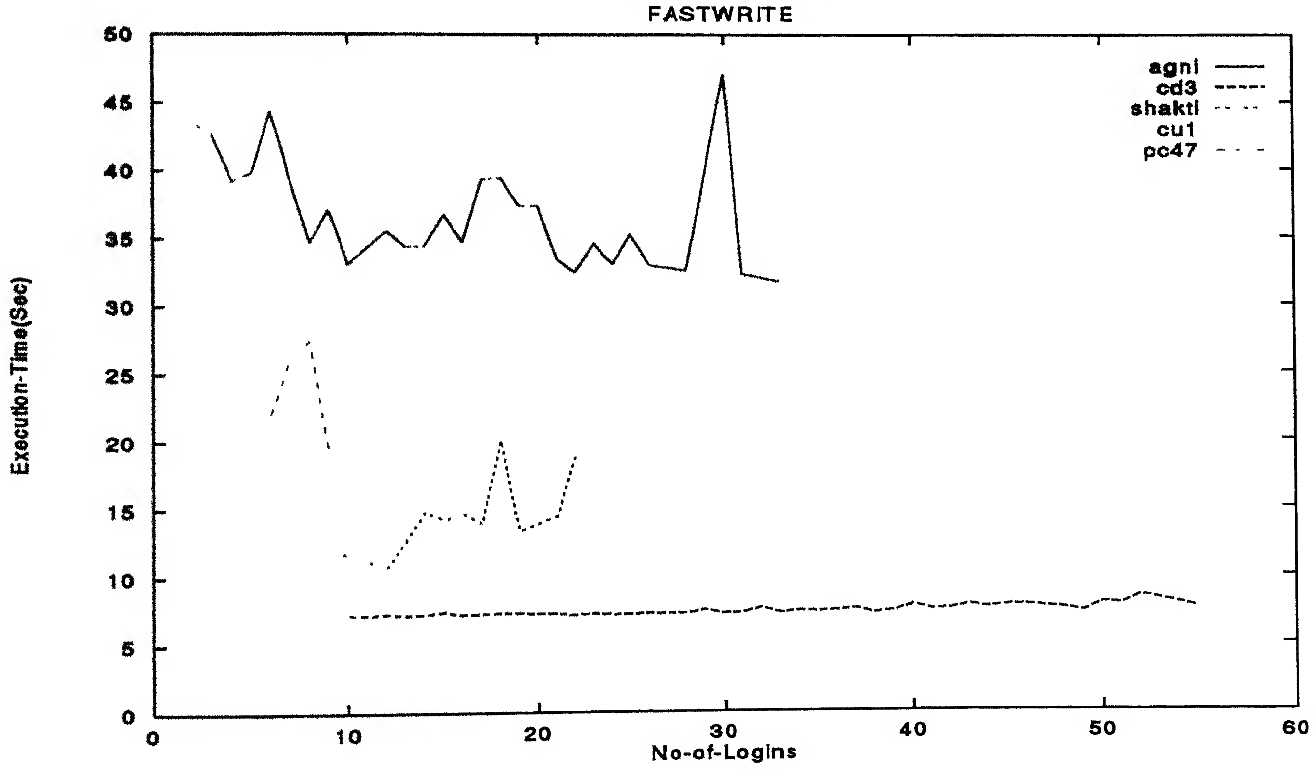
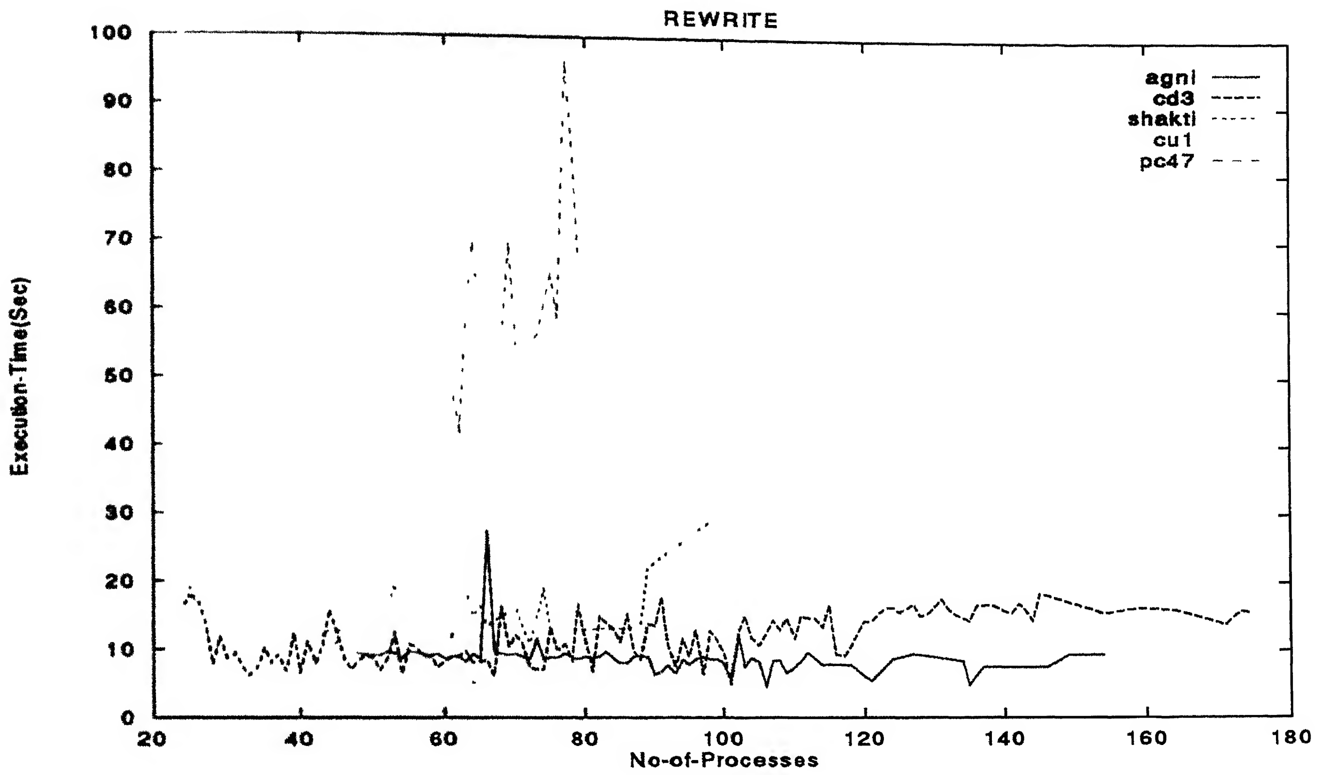


Figure 10: Fast Write Test

4.2.10 Rewrite Test

Each block of the file is read with the read system call, dirtied, and rewritten with write system call. Since no space allocation is done, and the I/O is well-localized, this should test the effectiveness of the filesystem cache and the speed of data transfer. Result graph is shown in figure 11. Following are the few observations from the graph.

- cu1 gives the best performance for this test. Which indicates that the file system of the cu1 is very efficient.
- sg2 is a little slower than cu1.
- agni improves a lot in this test.
- cd3 gives same performance as agni but it shows some variation with increase in load.
- shakti is also close to cd3 and agni but its performance goes slow with increasing load.
- pc47 is very poor as compared with other systems. Its performance goes even more down at higher load.



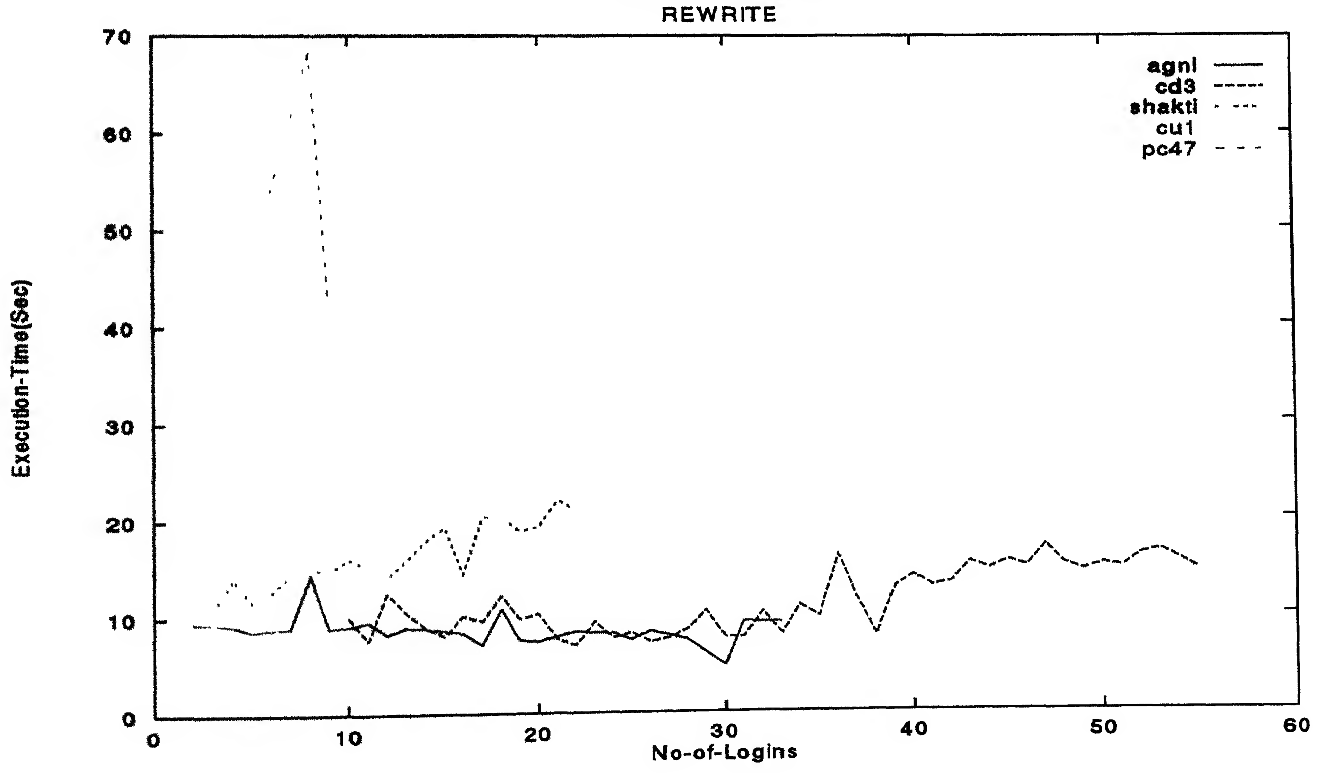
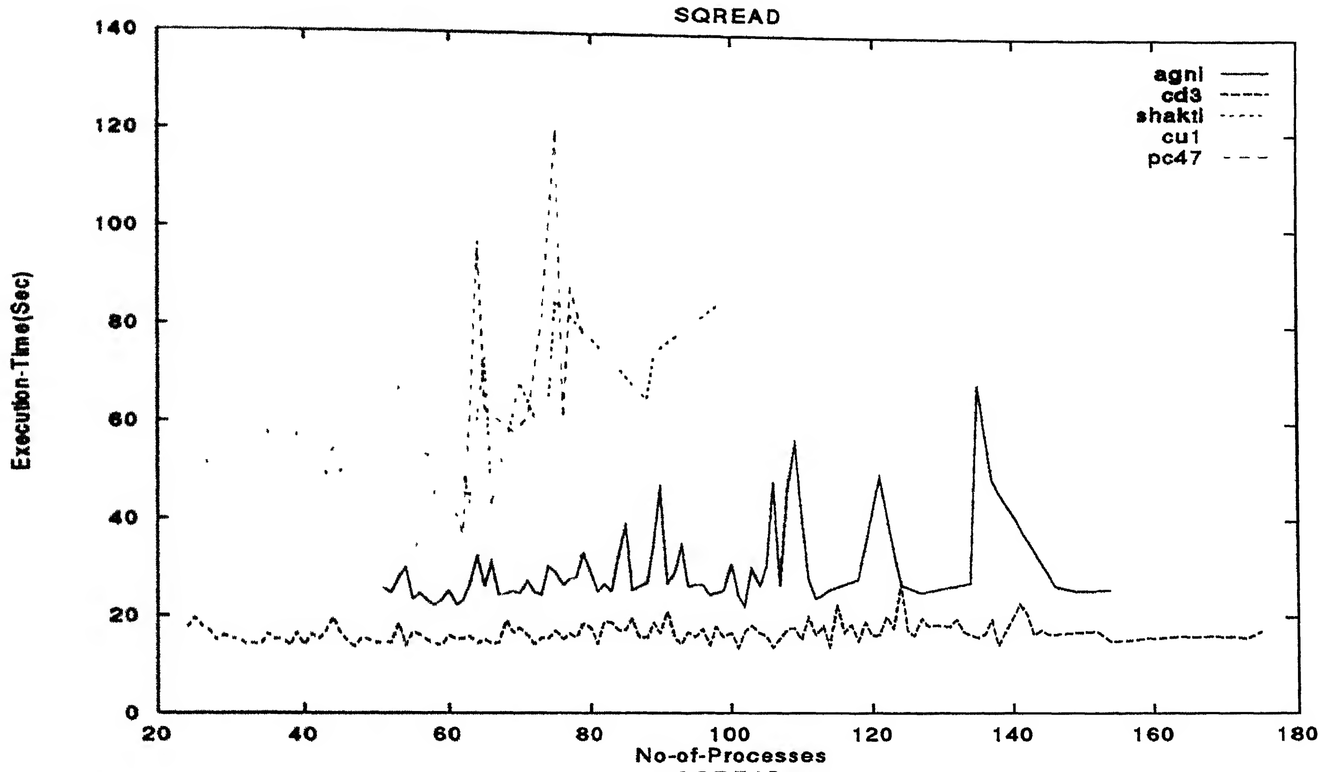


Figure 11: Re-write Test

4.2.11 Sequential Read Test

The file is read using the **getc()** stdio macro.This should check the performance of only stdio and sequential input operations. Figure 12 presents the graph for this test. Observations are summarized below.

- sg2 gives the best performance in per character sequential read too.
- cd3 is very close to sg2 and remain unaffected by the increasing load.
- agni is better in read operation than it was in write operation.
- cu1 is surprisingly slower than agni in this test.
- shakti is again slower than all other systems but pc47 and its performance goes down with increasing system load.
- pc47 is close to agni and cu1 at low load but it become extremely slow at high load.



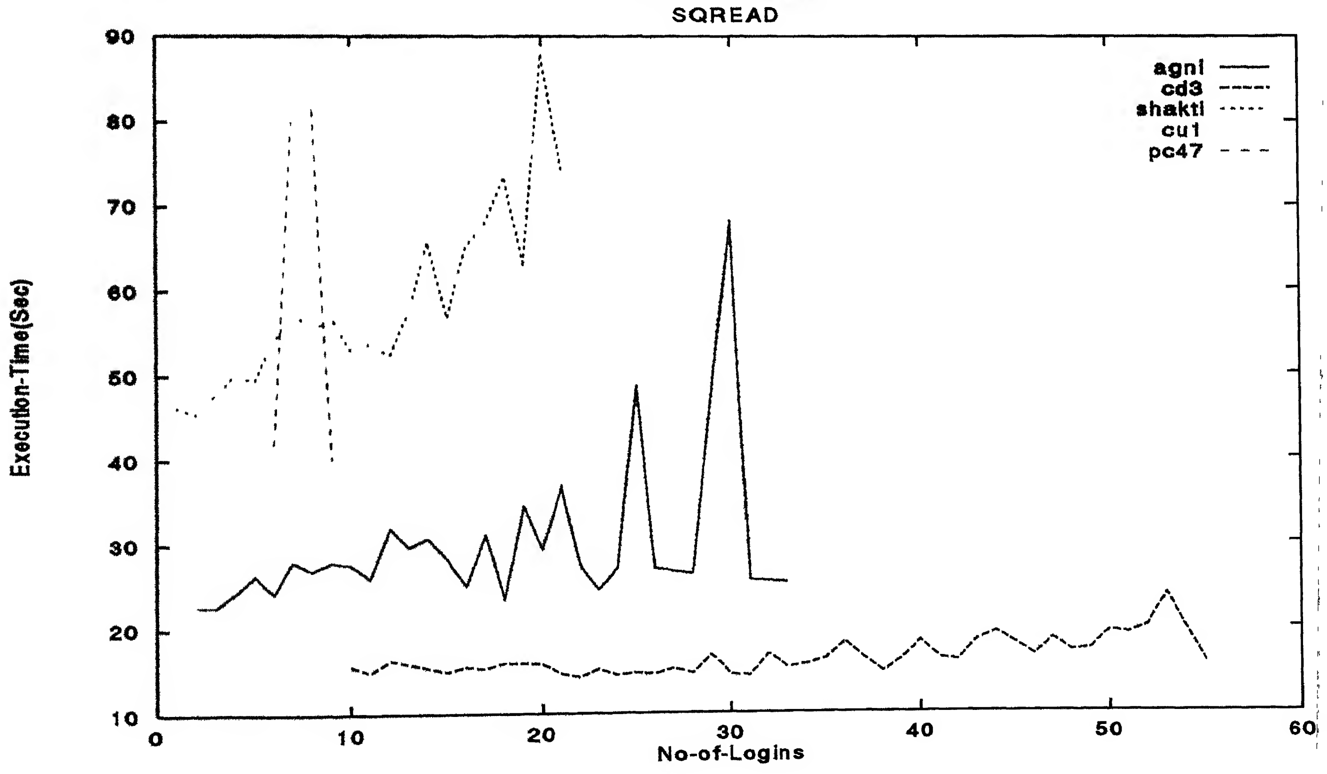
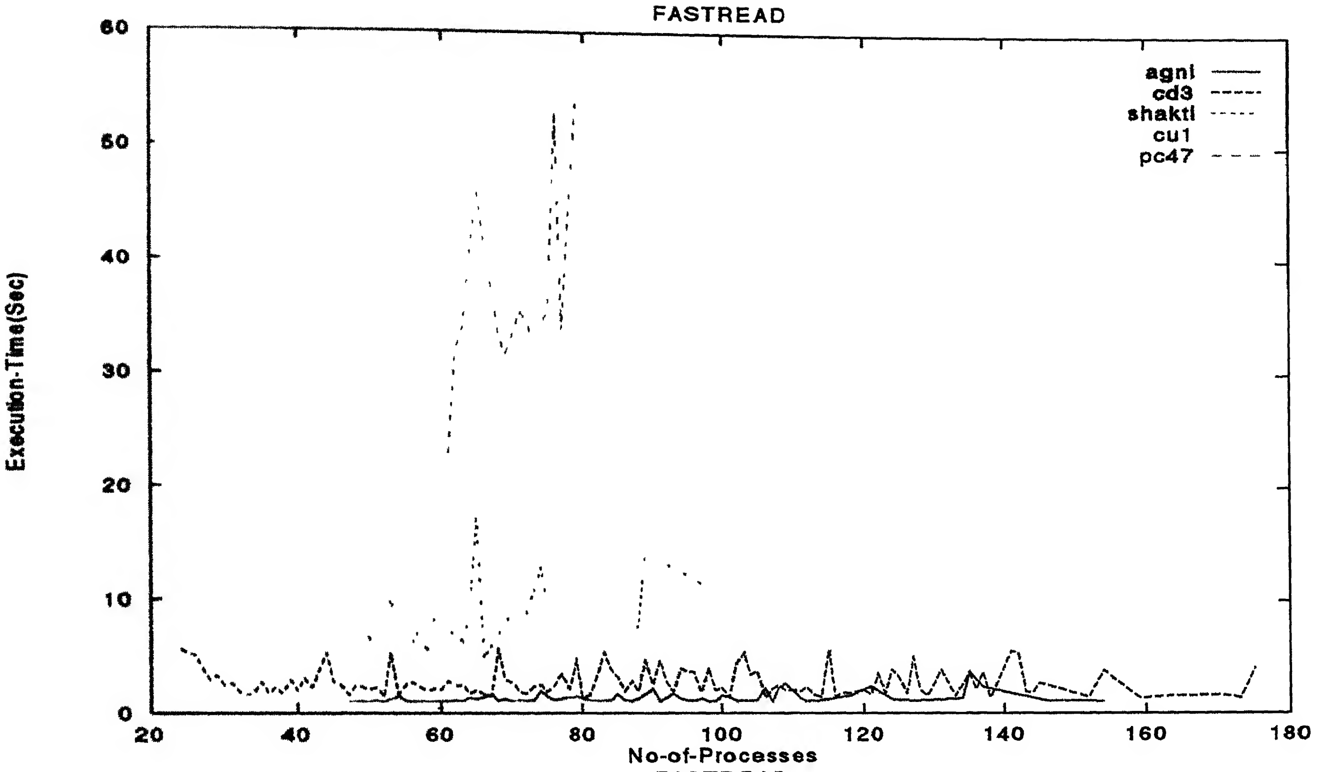


Figure 12: Sequential Read Test

4.2.12 Fast Read Test

The file is read using read system call. This should be a very pure test of sequential input performance. Figure 13 shows the result graph for this test. Points observed from the graph are listed below.

- cu1 is the fastest machine for this test.
- sg2 is very close to cu1 with a little fluctuation in performance.
- agni is as good as sg2,agni, and cu1. Its performance remains steady at the high load.
- cd3 is not very far from cu1, sg2 and agni but shows some fluctuation in performance.
- shakti is slower than cu1, sg2, agni and cd3 but this time the difference is not very large.
- pc47 is again very poor and its performance goes down at high system load.



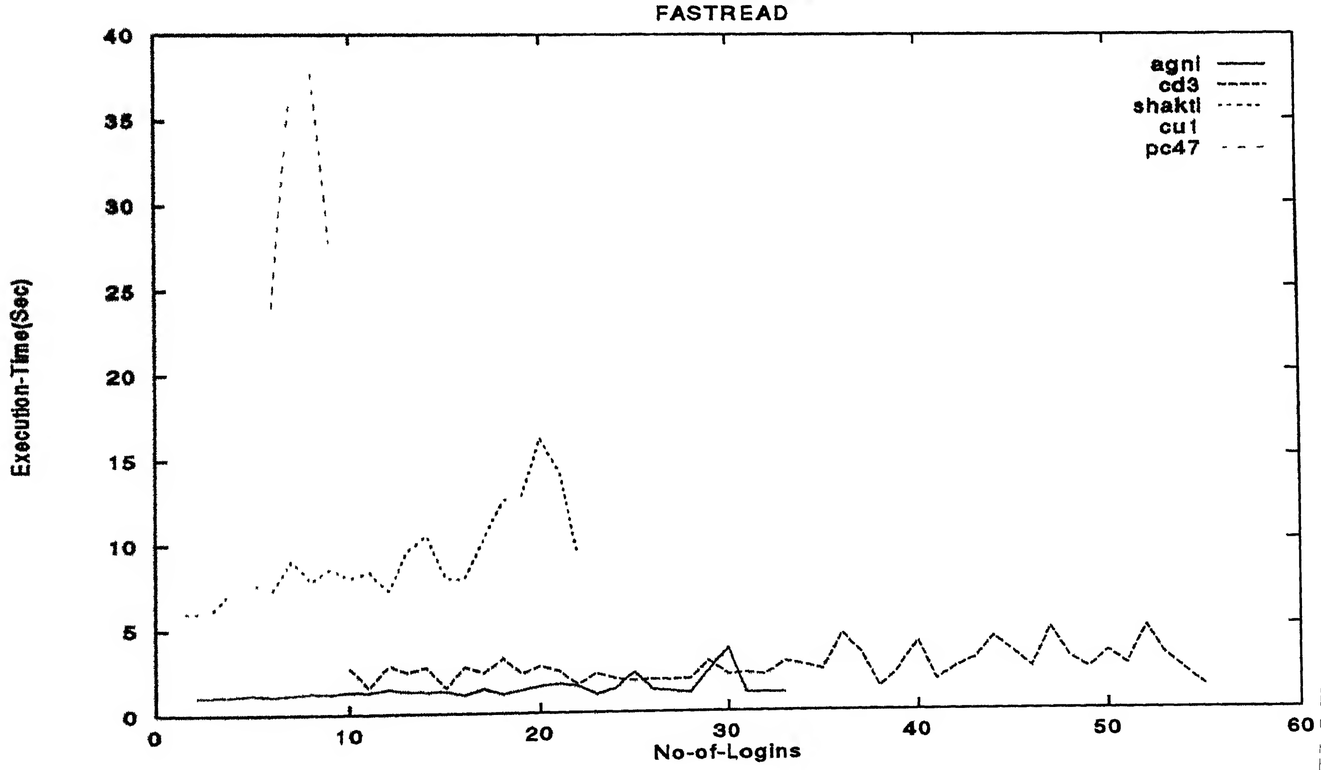


Figure 13: Fast Read Test

4.2.13 NFS Performance Tests

NFS performance tests were executed on pc47 after mounting the exported directories of remote systems through NFS. Five systems were compared in this test.

- sk3 directory of shakti.
- ag1 directory of agni.
- k1 directory of bhaskar (a HP 9000/819 system).
- ul4 directory of cu1.
- g21 directory of sg2.

The performance is shown against the number of bytes on the network in the last five minutes. It is observed that the network load does not affect NFS performance. It means that network bandwidth is not a bottleneck in NFS performance.

NFS Test

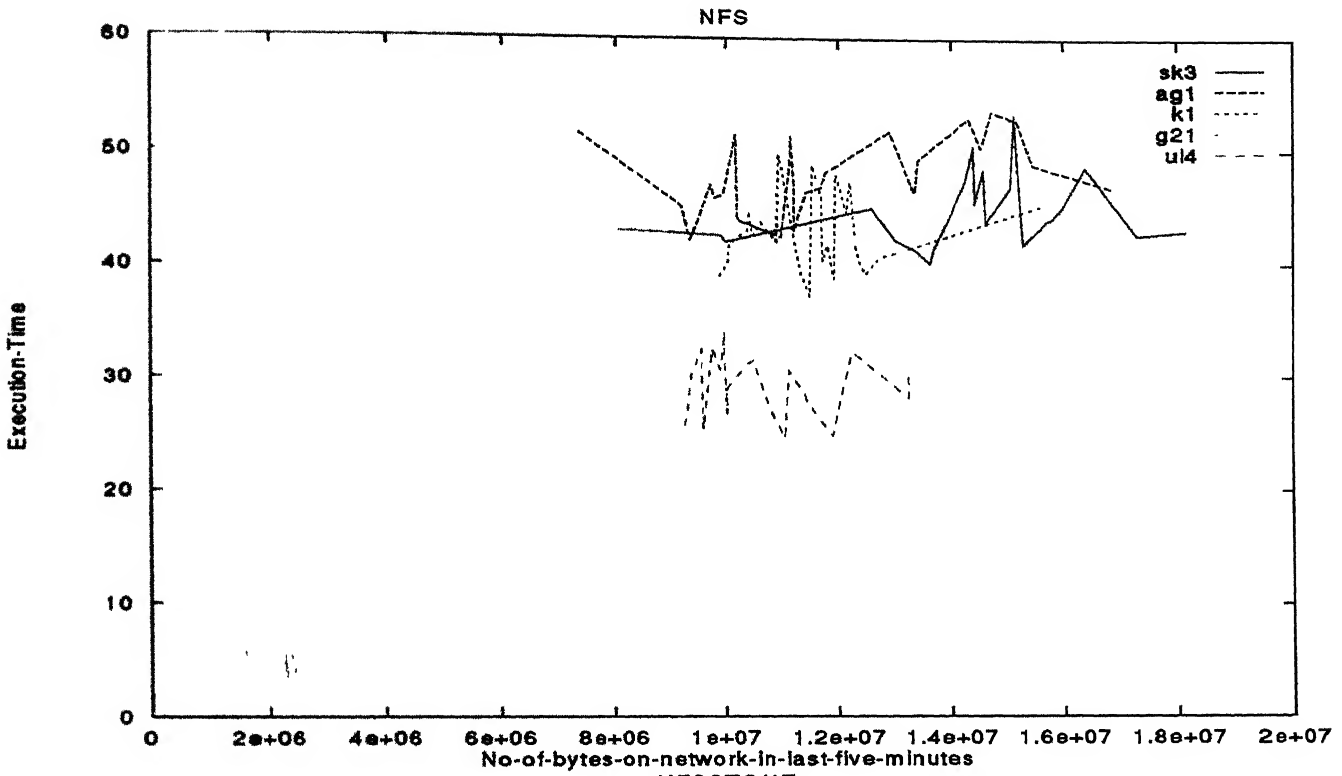
NFS test includes several NFS operations such as make dir, remove dir, copy, move, delete, create, read, write etc. The graph is shown in figure 14. Some observations are as follows.

- sg2 is the best NFS server available. Its performance is far more better than the other servers tested.
- cu1 is the second best NFS server in the lot but there is a large gap in the performance of cu1 and sg2.
- bhaskar, agni, and shakti are give nearly same performance. agni is a bit slow than other two.

▮ *NFSSTONE*

NFSstone produces results in NFSstones per second which is the direct measure of NFS performance. We should note that the system plotted higher on the graph gives the higher performance. Figure 14 presents the result graph for this test. Observations are listed below.

- sg2 is the best NFS server as rated by the NFS test.
- cu1 is better than all other systems but sg2.
- shakti, agni, and bhaskar are almost similar in performance but we can see that shakti is best among these three, second is agni and the slowest NFS server is bhaskar.



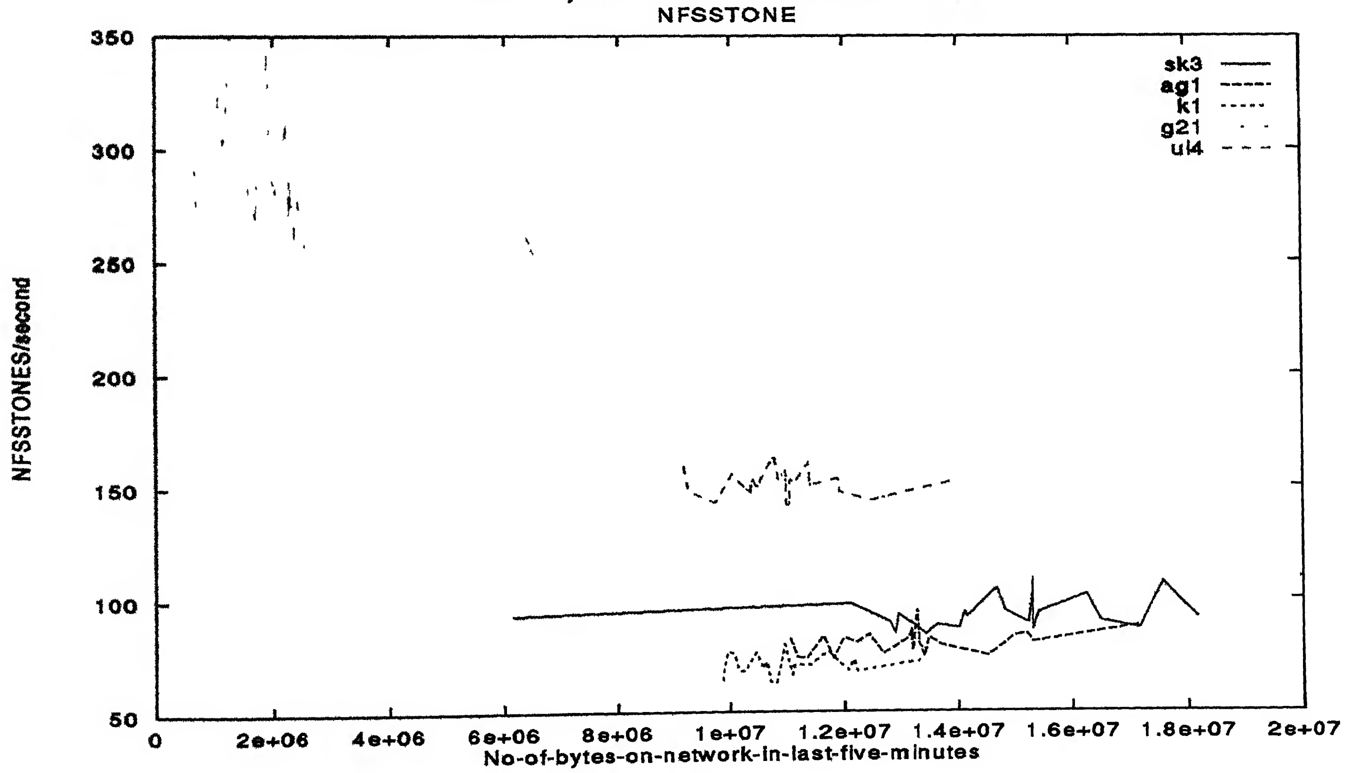


Figure 14: NFS Stone Tests

Chapter 5

Conclusions

A performance evaluation tool has been designed to evaluate the performance of computer systems in the IITK environment. The results can help a system administrator to assign the appropriate systems for different types of jobs. The tool can also be used to make purchase decisions for new machines and to find the bottlenecks in case of low performance. The tool includes a program to generate artificial load so that it can be used to evaluate the performance of a new system at the vendor's site.

The system load capturing routines record a number of things such as number of page faults, block input and output operations, processes waiting status etc. This information can be used to address various issues. Two such issues are the page fault behavior with increasing system load and the service bottleneck for waiting processes (such as network, disk I/O)

5.1 Summary of the Test Results

Several systems installed in Computer Center and the Department of Computer Science and Engineering were evaluated. Chapter 4 presents the result graphs which only includes the performance of six systems due to plotting limitations. We observe that the new Silicon Graphics machines are the fastest machines available in the Computer Center. SG machines show the best performance for almost all the test suites. DEC Alpha 2000 (dual processor) machine of CSE department is very close to the SG machines in performance. It was noted that the DEC Alpha 2000 machine shows steady performance at very high system load. Sun Ultra Sparc machine of CSE department is also a reasonably fast machine. The HP machines and single processor DEC Alpha systems installed in Computer Center are not good enough to be used as a server. Pentium PC running Linux is not a good machine for scientific computing or for using it as a server, but it is a good workstation for single user.

It was observed that some of the machines are heavily loaded and some machines are not utilized properly. SG machines are very powerful but they are not loaded properly. Some more users should be given accounts on the SG machines. Since these machines are good for scientific computing, the users who want to run long batch jobs should be given login on SG machines. Ultra Sparc system of CSE department is also very lightly loaded system. At the same time other servers of CSE department are heavily loaded. This power of this system should be utilized by running the server programs (e.g. NIS, DNS, mailserver etc.) on this system.

Another observation is that the systems with low CPU and disk I/O performance are used as the main NFS servers. It is believed that the network bandwidth is a bottleneck in the NFS performance. Test results show that the NFS performance is not affected by the network load. This means that the network is not saturated. Fast machines such as SG and Sun Ultra should be used as the main NFS servers.

5.2 Further Extensions

Work can be done to extend the scope of this tool. Following sections present some of the features that can be added in the present structure of this tool.

5.2.1 X-Server Performance Measurement

Some test programs can be designed to generate the workload for a X-Server. Some additional parameters should be defined in order to measure the current load on a X-Server. For instance, number of active windows, average number of mouse operations per unit time, mode (number of colors and resolution) of the X-Server etc. Another issue which matters is whether X-applications are running on the workstation or on some remote machines.

5.2.2 WWW Server Performance Measurement

The World Wide Web (WWW) is today's most popular and advanced information system. Most of the leading universities and their departments provide information related to admissions, courses, faculty profile, and ongoing research, on their web page. Selecting a good machine as web server and monitoring its performance has become important now. This tool can be extended to measure the WWW server performance by adding a program that generates http requests and records the response time. The machine running the test program and the machine running WWW server should be on the same LAN, as one is measuring the WWW server's performance and not the network bandwidth. Log of WWW server can be used to obtain the current load on the server.

5.2.3 Measuring the Performance of Multiprocessor Systems

A central issue in understanding performance measurement on multiprocessor architectures involves the pairing of architectures and applications. Some scientific programs requiring multiprocessor architecture should be included in the test suite to measure the performance of multiprocessor systems. The programs should be carefully selected as the suitability of code for the underlaying hardware and OS matters a lot in multiprocessors. [JLM88] and [MAMC86] are good texts available on this subject.

The test program not only measures the power of the processors but also the efficiency of the compiler and the operating system.

5.2.4 More on Network Performance

This tool does not examine the network performance in detail. The network load should be characterized to get the statistics of load generated by different applications using different protocols. Some early work in monitoring network traffic is discussed in [DEMK75]. A hierarchical approach is required to characterize the network load on different layers. For example in a network where different network layer protocols are being used, the first level can be to distinguish among the network layer protocols, such as, IP, ARP, ICMP (of TCP/IP suite), NetBios, IPX (of Novel), and NetBUI (of Microsoft). In the next level we can examine the IP packets for TCP or UDP. Further the packets can be examined for application layer protocols, such as, HTTP, FTP, TFTP, SMTP, NNTP, RPC, telnet, X-Server, YP, etc.

The fundamental issue in network performance measurement is throughput measurement [KO77]. However, the test programs used to generate the workload of different protocols should be used to measure the network performance in a more meaningful way. [HO86] presents an overview of performance analysis of LANs.

5.2.5 Measuring the Performance of Database Systems

Several ad-hoc benchmarks for database and transaction-processing systems exist. Most of the database and transaction-processing benchmarks measure the performance in vague metrics such as *transactions per second* and *queries processed per second*. Each "database product vendor" has implemented the standard ad-hoc benchmarks so that they can measure their product's performance against the other products. Work can be done to study transaction processing systems and to design a benchmark suite specifically for these systems. Some key issues in analyzing the database system architectures are discussed in [SByM87]. A comprehensive view of benchmarking for modern transaction processing and database systems is presented in [JG91].

Bibliography

[DEMK75] Dale P. Goodspeed David E. Morgan, Walter Banks and Richard Kolanko. A computer network monitoring system. *IEEE Transactions on Software Engineering*, SE-1(3):299–311, Sep 1975.

[Div96] Information Networks Division. Netperf: A network performance benchmark. *Hewlett Packard Company*, Feb 1996.

[Dix91] Kaivalya M. Dixit. The SPEC benchmarks. *Parallel Computing*, 17:1195–1209, 1991.

[Don87] Jack Dongarra. Computer benchmarking: paths and pitfalls. *IEEE Spectrum*, pages 38–43, July 1987.

[FAQ] Computer Benchmarks FAQ. URL. *http://sacam.oren.ortn.edu/ dave/ benchmark-faq.html.*

[Gaz] Linux Gazette. URL. *http://www.redhat.com/linux-info/lg/issue22/is sue22.html.*

[HO86] Joseph L. Hammond and Peter J.P. O'Reilly. *Performance Analysis Of Local Computer Networks*. Addition-Wesley Publishing Company, 1986.

[Hom] The Public Netperf Homepage. URL. *http://www.cup.hp.com/netperf /NetperfPage.html.*

[HOW] Linux Benchmarking HOWTO. URL. *http://sunsite.unc.edu/LDP /HOWTO/Benchmarking-HOWTO.html.*

[Jai91] Raj Jain. *The Art Of Computer Systems Performance Analysis.* John Wiley and Sons, INC, New York, 1991.

[JG91] Editor Jim Gray. *The Benchmark Handbook For Database and Transaction Processing Systems.* Morgan Kaufmann Publishers, San Mateo, CA, 1991.

[JLM88] Editor J. L. Martin. *Performance Evaluation of Supercomputers.* NORTH-HOLLAND, 1988.

[KO77] Leonard Kleinrock and Holger Opderbeck. Throghput in the ARPANET-protocols and measurement. *IEEE Transactions on Communications*, COM-25(1):95–104, Jan 1977.

[MAMC86] G. Balbo M. Ajmone Marsan and G. Conte. *Performance Models of Multiprocessor Systems.* The MIT Press, 1986.

[MBL91] G. Cybenko M. Berry and J. Larson. Scientific benchmark characterizations. *Parallel Computing*, 17:1173–1194, 1991.

[McK88] Phillip McKerrow. *Performance Measurement of Computer Systems.* Addition-Wesley Publishing Company, 1988.

[NB75] Editor Nicholas Benwell. *BENCHMARKING: Computer Evaluation and Measurement.* Hemisphere Publishing Corporation, 1975.

[Org] The SPEC Organization. URL. *http://www.specbench.org/spec.*

[Pri89] Walter J. Price. A benchmark tutorial. *IEEE Micro*, pages 28–43, Oct 1989.

[Rao97] P. V. Nageswara Rao. NETMON: A network monitoring tool. Master's thesis, Indian Institute Of Technology Kanpur, India, 1997.

[Sax93] Navin Kumar Saxena. A NFS server for DOS. Master's thesis, Indian Institute Of Technology Kanpur, India, July 1993.

[SByM87] Alan R. Hevner S. Bing yao and Helene Young Myers. Analysis of database system architecture using benchmarks. *IEEE Transactions on Software Engineering*, SE-13(6):709–725, June 1987.

[Ser] The Performance Database Server. URL. *http://performance.netlib .org/performance/html/PDSpapers.html.*

[TPC] TPC. URL. *http://www.tpc.org.*

[vdS91] Aad J. van der Steen. The benchmark of the euroBen group. *Parallel Computing*, 17:1211 1221, 1991.

[Ver] Linpack Benchmark Java Version. URL. *http://www.netlib.org/benc hmark/linpackjava.*

[Wei91] Reinhold P. Weicker. A detailed look at some popular benchmarks. *Parallel Computing*, 17:1153–1172, 1991.